PADERBORN
UNIVERSITY

Distributed Algorithms for Modern Communication Networks

A PhD Thesis

In partial fulfillment
of the requirements for
the academic degree

Doctor rerum naturalium (Dr. rer. nat.)

submitted to the

Faculty of Computer Science, Electrical Engineering and Mathematics
- Department of Computer Science -
Paderborn University

by

Thorsten Götte

Reviewers:	Prof. Dr. Christian Scheideler
	Prof. Dr. Sevag Gharibian

Abstract

This thesis considers algorithmic solutions to problems that arise in the theoretical study of modern communication networks, such as the Internet. The thesis is divided into two main parts. In the first part of this thesis, we show how to construct an overlay network of degree $O(\log n)$ and diameter $O(\log n)$ in $O(\log n)$ time starting from an arbitrary weakly connected graph. We assume a synchronous communication network in which nodes can send messages to nodes they know the identifier of, and new connections can be established by sending node identifiers. If the initial network's graph is weakly connected and has constant degree, then our algorithm constructs the desired topology with each node sending and receiving only $O(\log n)$ messages in each round in time $O(\log n)$, w.h.p. Since the problem cannot be solved faster than by using pointer jumping for $O(\log n)$ rounds (which would even require each node to communicate $\Omega(n)$ bits), our algorithm is asymptotically optimal. Additionally, we show how our algorithm can be used to efficiently solve graph problems in the novel HYBRID model. Motivated by the idea that nodes possess two different modes of communication, we assume that communication of the *initial* edges is unrestricted, whereas only polylogarithmically many messages can be sent over edges that have been established throughout an algorithm's execution. For an (undirected) graph G with arbitrary degree, we show how to compute connected components and a spanning forest in $O(\log n)$ time, w.h.p. Furthermore, we show how to compute an MIS in time $O(\log \Delta + \log \log n)$, w.h.p., where Δ is the initial degree of G.
In the second part of the thesis, we consider the problems of computing compact routing tables and low-diameter decompositions for a (weighted) graph $G := (V, E, \ell)$ that can be separated through k shortest paths. This class of graphs includes planar graphs, graphs of bounded treewidth, and graphs that exclude a fixed minor K_r. We present algorithms in the CONGEST and the novel HYBRID communication model that are competitive in all relevant parameters:

- For a given parameter $\epsilon > 0$, we compute a routing scheme with stretch $1 + \epsilon$. Our scheme computes labels of size $\widetilde{O}(k\epsilon^{-2})$ and is computed in $\widetilde{O}(k\epsilon^{-3})$ time in the HYBRID model and $\widetilde{O}(k\epsilon^{-3} \cdot \text{HD})$ time in CONGEST. Here, HD denotes the network's hop diameter.
- Given a parameter $\mathcal{D} > 0$, our low-diameter decomposition algorithm divides the graph into connected subgraphs of strong diameter $\mathcal{D}$. An edge $e \in E$ of length ℓ_e has endpoints in different subgraphs with probability $O(\ell_e \cdot \log(k \log n)/\mathcal{D})$. The decomposition can be computed in $\widetilde{O}(k)$ time in the HYBRID model and $\tilde{O}(kHD)$ time in CONGEST.

Broadly speaking, we present distributed and parallel implementations of sequential divide-and-conquer algorithms where we replace exact shortest paths with approximate shortest paths. In contrast to exact paths, these can be efficiently computed in the distributed and parallel setting. Further, and perhaps more importantly, we show that instead of explicitly computing vertex-separators to enable efficient parallelization of these algorithms, it suffices to sample a few random paths of bounded length and the nodes close to them. Finally, we present a SetCover algorithm in the Beeping model. Our algorithm runs in $O(k^3)$ time and has an expected approximation ratio of $O(\Delta^{3/k} \log^2 \Delta)$. The value $k \in [3, \log \Delta]$ is a parameter that lets us trade runtime for approximation ratio similar to the celebrated algorithm by Kuhn and Wattenhofer [PODC '03]. This algorithm can then be extended to effieciently solve the DominatingSet-problem in a graph's $\mathcal{D}$-neigborhood in a distributed fashion.

Zusammenfassung

In dieser Arbeit betrachten wir algorithmische Lösungen für fundamentale Probleme in modernen Kommunikationsnetzen. Im ersten Teil dieser Arbeit zeigen wir, wie man ein Overlay-Netzwerk mit Grades und Durchmesser $O(\log n)$ in $O(\log n)$ Runden ausgehend von einem beliebigen, schwach verbundenen Graphen konstruiert. Wir gehen von einem synchronen Kommunikationsnetz aus, in dem Knoten Nachrichten an alle Knoten senden können, deren Adresse sie kennen, und neue Verbindungen durch das Versenden dieser Adressen hergestellt werden können. Wenn der Ausgangsgraph des Netzwerks schwach zusammenhängend ist und einen konstanten Grad hat, dann konstruiert unser Algorithmus die gewünschte Topologie mit hoher Wahrscheinlichkeit in $O(\log n)$ Runden, wobei in jeder Runde nur $O(\log n)$ Nachrichten sendet und empfängt. Da das Problem nicht schneller gelöst werden kann als durch sogenanntes *Pointer Doubling* für $O(\log n)$ Runden (was sogar erfordern würde, dass jeder Knoten $\Omega(n)$ Bits kommuniziert), ist unser Algorithmus asymptotisch optimal. Außerdem zeigen wir, wie unser Algorithmus zur effizienten Lösung von Graphenproblemen im HYBRID model verwendet werden kann. Motiviert durch die Idee, dass Knoten zwei verschiedene Arten der Kommunikation besitzen, nehmen wir an, dass die Kommunikation der Kanten unbeschränkt ist, während nur polylogarithmisch viele Nachrichten über Kanten gesendet werden können, die während der Ausführung eines Algorithmus etabliert wurden. Für einen (ungerichteten) Graphen G mit beliebigem Grad zeigen wir, wie man zusammenhängende Komponenten und einen Spannwald mit hoher Wahrscheinlichkeit in $O(\log n)$ Zeit berechnen kann. Außerdem zeigen wir, wie man ein MIS mit hoher Wahrscheinlichkeit in $O(\log \Delta + \log\log n)$, berechnet, wobei Δ der Grad von G ist.

Im zweiten Teil der Arbeit betrachten wir das Problem der Berechnung von kompakten Routing-Tabellen und Dekompositionen mit geringem Durchmesser für einen (gewichteten) Graphen $G := (V, E, \ell)$, der durch k kürzeste Wege separiert werden kann. Zu dieser Klasse von Graphen gehören planare Graphen, Graphen mit beschränkter Treewidth und Graphen, die einen festen Minor K_r ausschließen. Wir präsentieren Algorithmen im CONGEST- und im neuartigen HYBRID-Kommunikationsmodell, die in allen relevanten Parametern konkurrenzfähig sind:

- Für einen gegebenen Parameter $\epsilon > 0$ berechnen wir ein Routing-Schema mit Stretch $1+\epsilon$. Unser Schema berechnet Label der Größe $\widetilde{O}(k\epsilon^{-2})$ und wird in $\widetilde{O}(k\epsilon^{-3})$ Zeit im HYBRID-Modell, und $\widetilde{O}(k\epsilon^{-3} \cdot \mathrm{HD})$ Zeit in CONGEST. Dabei bezeichnet HD den Hopdurchmesser des Graphen.
- Für einem Parameter $\mathcal{D} > 0$ unterteilt unser Algorithmus zur Dekomposition den Graphen in zusammenhängende Subgraphen mit starkem Durchmesser $\mathcal{D}$. Eine Kante $e \in E$ der Länge ℓ_e hat ihre Endpunkte in zwei verschieden Subgraphen mit der Wahrscheinlichkeit $O(\ell_e \cdot \log(k \log n)/\mathcal{D})$. Die Dekomposition kann in $\widetilde{O}(k)$ Zeit im HYBRID-Modell und $\tilde{O}(kHD)$ Zeit in CONGEST berechnet werden.

Wir stellen verteilte und parallele Implementierungen von sequenziellen Divide-and-Conquer-Algorithmen vor, bei denen wir exakte kürzeste Pfade durch approximative kürzeste Pfade ersetzen. Im Gegensatz zu exakten Pfaden können diese in der verteilten und parallelen Umgebung effizient berechnet werden. Außerdem, zeigen wir, dass es ausreicht, anstelle der expliziten Berechnung von Vertex-Separatoren, einige zufällige Pfade begrenzter Länge zu wählen und die Separatoren um diese herum zu konstruieren. Schließlich stellen wir einen SetCover-Algorithmus für das Beeping-Modell vor. Unser Algorithmus läuft in $O(k^3)$ Zeit und hat eine erwartete Approximationsgüte von $O(\Delta^{3/k} \log^2 \Delta)$. Der Wert $k \in [3, \log \Delta]$ ist ein Parameter, mit dem wir Laufzeit gegen Approximationsgüte eintauschen können, ähnlich wie bei dem Algorithmus von Kuhn und Wattenhofer [PODC '03]. Dieser Algorithmus kann erweitert werden, um das DominatingSet-Problem in der $\mathcal{D}$-Nachbarschaft eines Graphen mittles eines verteiltes Algorithmus effizient zu lösen.

Für Oma Röse und Oma Tilla.

Acknowledgments

Es ist vielen großartigen Menschen zu verdanken, dass es diese Dissertation gibt. Zu allererst möchte ich Christian Scheideler danken. Du hattest stets großes Vertrauen in meine Arbeit und ein offenes Ohr für Fragen. Vielen Dank, dass Du mir die diese Promotion ermöglicht hast. Weiterhin möchte allen aktuellen und ehemaligen Mitgliedern der Fachgruppe danken: Robert, Thim, Kristian, Alexander, Michael, Christina, Ardalan, Daniel, Andreas, Julian, David, Jinfeng, Jelly, Jonas, Jan-Luca und Andre sowie natürlich auch Mafi, Uli, Tici, Marion, Petra, Rainer und Ulf. In der Gruppe war stets eine tolle Atmosphäre, sowohl bei der Zusammarbeit in der Lehre als auch im personlichen Miteinander. Ich werde immer mit Freude auf die Zeit mit Euch zurückblicken. Insbesondere möchte ich Alexander, der meine Masterarbeit betreut und mich zur Promotion motiviert hat, sowie und meinen Bürokollegen Kristian und Jinfeng für die vielen tollen Gespräche über Gott und die Welt danken (感谢你们所做的一切). Außerdem möchte ich hervorheben, dass ich ohne unsere Sekretärin Marion vermutlich an meiner eigenen Desorganisation gescheitert wäre. Als nächstes möchte ich allen meinen Koautoren Vipin, Kristian, Michael, Christina, Julian, Henning, Seth, John, Vijeth und Jinfeng für die tolle und anregende Zusammenarbeit danken. Die Paper, welche die Grundlage dieser Arbeit bilden, gäbe es ohne Euch nicht. Thank you for everything! Zuletzt möchte ich meinen Freunden, Geschwistern und allen voran meinen Eltern danken. Ohne euer Vertrauen und eure Unterstützung hätte ich das nicht geschafft.

Thorsten Götte

Contents

List of Figures

List of Tables

Jetzt mal Butter bei die Fische.
(Let's butter them fishes.)

Northern German Saying

1 Introduction

In this thesis, we will consider a variety of algorithmic problems that arise in the context of what we call *modern communication networks*. Making such a general statement, of course, raises several questions: What precisely *is* a modern communication network? And, perhaps more importantly, why should we care? Let us begin this quest by trying to characterize communication networks in general. From the hardware perspective, a communication network is an interconnected network of electronic devices where each device is physically connected to at least one other device, e.g., by a cable or a wireless connection. Being connected to the network enables the sending of electronic messages to other connected devices. In a *modern* network, possible devices range from classical web servers and personal computers to mobile handheld devices, sensor nodes that monitor their environment, and even everyday household appliances like televisions and fridges. Given this area's ongoing rapid technical advancement, this enumeration is far from exhaustive. Due to their widespread availability, communication networks become ubiquitous in nearly anyone's professional and private life. In fact, there is a non-trivial probability that you, the reader, have obtained this thesis using a communication network, namely the Internet. The Internet is undoubtedly the largest communication network on the planet and, therefore, a prime example of a modern network. That being said, reducing the Internet's capabilities to just point-to-point communication seems overly simplifying. Numerous examples of more advanced applications built upon the Internet's communication capabilities are easily available. *Social media* platforms allow individuals to connect with friends and family, while online forums foster discussions and collaborations on diverse topics. *File sharing* services permit us to find and exchange large data items between internet users easily. *E-commerce* enables people to buy and sell goods and services online easily. *Online banking* or *Cryptocurren-*

cies facilitate secure financial transactions between (possibly anonymous) parties. As with devices, this list of possible applications is far from complete.

However, a communication network is not only defined by its hardware. To put it bluntly, we cannot hope to obtain an efficient network by merely linking up some devices. The physical network connections only ensure that *in principle* any device connected can communicate with any other device. In addition to these physical aspects, a suite of protocols and algorithms is required to ensure that any device can efficiently communicate. To illustrate this, consider road traffic as an analogy and think of these physical connections as roads and cars as messages. Physical connections only ensure a series of roads to a destination *exists*. But this fact alone does not guarantee one reaches it safely and timely. There is a variety of additional mechanisms in place to ensure this. Laws and traffic rules govern the right of way and ensure safety. Maintenance crews remove obstructions like fallen trees. Traffic controllers monitor the congestion and bottlenecks and declare detours around damaged roads. Without such checks and balances, there is a high chance of getting stuck (at a dead end) or, even worse, having an accident (with a deadly end). In the context of the Internet, the equivalents of these tasks are fulfilled by distributed algorithms and protocols. Currently, there are around 500 different protocols and technical standards that ensure the Internet's basic functionalities. As networks only continue to growing bigger in both size and socio-economic importance, there is an ever increasing need for efficient an scalable algorithms. With this thesis, we hope to make a humble contribution towards this.

1.1 Contributions & Structure of this Thesis

Having introduced the general setting, the obvious question is what we're doing in this thesis. Or, to put it differently, what challenges do modern communication networks face? The contributions of this thesis are divided into two parts. Before we take a deep dive into the details of this thesis' contributions in their respective parts, let us introduce the challenges and our solutions on a high level. In the following, we will informally state the main takeaways and discuss their implications. We will make a formal statement of precise contributions later after we have established the concrete models and problem definitions. Here, we are interested in a high-level classification and consciously omit some details for an easier presentation and understanding.

After that, in the remainder of the introduction, we establish some preliminaries. In Section 1.2, we establish the theoretical background that is needed for this thesis. This includes some basic notions from graph theory, topology, complexity and probability theory. After that, we present all models that we are going to use in this thesis in Section 1.3. This includes the classical CONGEST model of distributed computing as well as more recent models that better resemble the capabilities of modern communication networks. Finally, in Section 1.4, we present a list of publications the author of this thesis was involved in.

1.1.1 Part I: Fast Construction of Overlays and its Applications

Many modern distributed systems (especially those that operate via the Internet) are not concerned with the physical infrastructure of the underlying network. Instead, these large-scale distributed systems form *logical networks* that are often referred to as *overlay networks*, *peer-to-peer networks*, or simply *overlay*. Note that every device connected to the Internet is assigned to a unique *IP address*. In an overlay, two devices are considered as *connected* if they know each other's IP addresses. The underlying network ensures that a message for a given address will eventually reach the corresponding device. Both the sender and the receiver remain completely unaware of the message's actual path in the underlying network.

There are many practical benefits of this approach. First, it simplifies the design of algorithms and protocols, as the designers don't need to consider the minutiae of the underlying network. Furthermore, since the protocols are agnostic of their underlying infrastructure, the overlay network can be extended to devices that use different physical carriers. Thus, while the physical network(s) interconnecting the network members are highly heterogeneous, the overlay appears homogeneous. This tremendously simplifies the design of distributed applications. Furthermore, as all connections in an overlay are purely virtual, they are not static but can be created at will. One simply needs to know its address to connect to another network member. This allows the construction of overlay topologies tailored to the specific application.

Overlay networks surged in popularity in the early 2000s as the Internet became more widely available to the public. In practice, they were mainly used to (often illegally) share large files like movies or music between end users through platforms like *Napster*[Wik24d] or *KaZaa* [Wik24b]. In more recent years, they have been used as the basis for cryptocurrencies like *Bitcoin* [Nak08] and *Ethereum* [But14]. Also, the microblogging service *Mastodon*[Wik24c], an alternative to the (as of the time of this writing) most popular service $\mathbb{X}$, has a decentralized architecture that uses an overlay network. A common denominator in all these real-world applications is their openness. All these platforms want to make it easy for new users to join and allow them to exchange files, money, and opinions free of any restrictions. Thus, everyone and their mother can join these networks provided they have Internet access. As ever so often, this noble design philosophy comes with several downsides. First off, it raises several legal and ethical questions. Napster and KaZaa were shut down due to massive Copyright infringements, cryptocurrencies are often used to finance illicit activities, and Mastodon is unable to efficiently moderate right-wing propaganda. However, deciding whether or not these matters are necessary evils is clearly beyond the scope of this thesis. We are interested in the technical implications of a lack of admission control on the construction and maintenance of overlays. Allowing *everyone* to join the network unconditionally causes a lot of fluctuation in the network membership, so-called *churn*. New devices may join the network, while older devices may leave or fail. In fact, studies on popular peer-to-peer networks have shown [SR06] that about 50% of the devices leave the network within an hour but are *replaced* by roughly the same number of new devices. This emphasizes how volatile these networks can be.

Further, as all devices are exposed by their IP addresses, they can be subject to DDoS attacks: An attacker could flood a critical device with requests until it shuts down. In addition to all this, attacks or malfunctions in the physical infrastructure make certain devices unreachable. To summarize, a plethora of factors outside of the network's control can cause it to be in an arbitrary state. In order to retain its functionality, it must be able to *recover* from these states on its own. By *recover*, we mean that the network returns to a well-connected overlay. Thus, distributed algorithms that can construct a desired overlay from any arbitrary initial overlay are desperately needed. These algorithms can be executed after transient failures or large-scale attacks or simply permanently run in the background to maintain the overlay. Therefore, we ask ourselves the following question:

Challenge 1: *Construction of Overlay Networks*

How can we efficiently construct a well-connected overlay network starting from an arbitrary (connected) configuration?

A naive algorithm would let each device continuously introduce all its known devices to one another. Suppose we have n network devices, and each device stores the IP address of, say, Δ other devices. The naive solution sends each known address to the other $\Delta - 1$ devices. Afterward, each device knows $\Delta(\Delta - 1)$ addresses, and the process is repeated. It is easy to see that all addresses are known to all devices after $O(\log n)$ iterations. This is optimal with respect to time. However, this approach requires each device to send $O(n)$ messages eventually. This is too much if we consider devices with limited networking capabilities. As the first main contribution of this thesis, we show that perhaps one of the simplest algorithms imaginable is surprisingly worst-case optimal. Instead of exhaustively sending *all* known addresses to *all* known devices, we take a more cautious approach. On a high level, we send each known address to *one* device that we pick independently and uniformly at random. Thereby, the addresses perform a random walk in the overlay network. This ensures that each device always sends and receives only $O(\Delta)$ addresses on expectation, and eventually, all the addresses will distributed uniformly at random. However, these walks would take a long time to reach a uniform distribution in an arbitrary, badly connected overlay. To mitigate this, each device periodically connects to d of its known addresses, creating a new (slightly better connected) overlay. After $O(\log n)$ repetitions, the emerging overlay is extremely well-connected despite each device only knowing Δ other devices.

Given this intuitive description of the algorithm, we summarize the main takeaway from the first part of this thesis as follows:

Takeaway 1

We can construct overlay networks fast and efficiently through simple random walks!

Of course, we omitted some details in the description of the algorithm. The algorithms' exact descriptions and their analysis can be found in Part I. Therein, we also present solutions to simple algorithmic problems that can be efficiently solved with our overlay construction algorithm as a subroutine. These problems include the fast construction of spanning forests and (in the author's opinion rather surprisingly) also an improved algorithm for the well-known Maximal Independent Problem. These applications emphasize that the power of non-local communication compared to traditional models of communication where nodes can only communicate with their immediate neighbors. We explore this further in the second part of the thesis.

1.1.2 Part II: Distributed Algorithms for Graph Problems

For all their benefits, the overlays mentioned above are built under one critical assumption: the underlying network provides reliable and efficient point-to-point messaging. That means, given some abstract *address* of another member in the network, one can send a message that reaches this member. As stated before, such efficient communication between multiple parties is an (if not *the* most) integral functionality of a communication network. It is ensured by so-called *routing schemes*, distributed algorithms that control the forwarding of packets from one device to another. Given a packet with sender s and a receiver t, a routing scheme chooses the path that the packet takes from s to t.

There is a plethora of measures for the quality of routing paths. The most prominent ones either limit the number of routing paths that contain a given node, i.e., they limit the so-called *congestion*, or the length of the routing paths compared to the graph's shortest path metric. In this thesis, we are only interested in the latter. With access to the network's complete topology, the problem of finding the shortest path between all its members is well-understood. If all connections are known, there are polynomial time algorithms for this problem, e.g., the famous algorithm of Floyd and Warshall [Flo62] that computes the shortest paths between all pairs of nodes in a graph (APSP). This would immediately give us the best possible routing path from each device to each other device. However, these exact paths do not (necessarily) result in an efficient routing scheme. If each device learns all distances to *all* other devices in the network, it needs to store routing information for n paths, which may require the device to store $O(n)$ bits. This is not feasible for large networks as n grows too big. The crux of most routing schemes is to limit the information each device has stored. To this end, we consider so-called *low-stretch* routing paths. These paths may be longer than that than the exact shortest path by a predefined factor called *stretch*. This gives us some slack in the computation as we are not required to find the exact shortest paths but only *good* enough paths that can be computed faster and require the nodes to store less routing information. Often, such routing schemes are called *compact* routing schemes.

But where and how are these routing schemes computed? A simple strawman solution would be to elect a single device, let's call it the *controller*, that gathers all information on the network's topology and then locally computes the optimal routing paths. Then, it sends the necessary routing information back to the devices. While using a central controller might be a worthwhile solution for *small* networks in practice, this approach does not scale well to larger networks. In a large network, node crashes, edge failures, and benign changes such as new members joining and old members leaving are virtually unavoidable. Therefore, the routing scheme must be recomputed frequently to adapt to the changing network. Consequently, the controller must always have complete, up-to-date knowledge of the network. Such a design would be impractical for many reasons. First, continuously monitoring the network and feeding the state to the controller requires a lot of processing power and bandwidth from the controller. This does not scale for large networks like the Internet, as the number of devices and connections is too large. Furthermore, such a controller would be a single point of failure. That means it cannot execute the algorithm and compute new paths if it crashes under the load, becomes corrupted due to an attack, or is otherwise malfunctioning. For these reasons, it is desirable to have a completely decentralized approach to computing the routing paths. At least as a fallback mechanism if the controller is unavailable for a certain time frame. This means no single controller should compute the paths but a collaborative mechanism, i.e., a distributed algorithm involving all the members.

This leads us to the following challenge that we are going to address in the second part of the thesis:

Challenge 2: *Construction Of Routing Schemes*

How can we efficiently construct a low-stretch routing scheme in a distributed manner?

In [EN18b], Elkin and Neimann essentially settled the problem for general network topologies, i.e., without restricting how the devices may be interconnected. They present a routing scheme with stretch $O(k)$ that requires each device to store $O(n^{\frac{1}{k}} \operatorname{polylog} n)$ bits of routing information that can be efficiently computed in a distributed manner. However, if one designs an algorithm for any topology, that algorithm has to consider pathologic worst-case topologies. There are network topologies where a routing scheme with stretch k necessarily requires each device to store $\Omega(n^{\frac{1}{k}})$ bits of routing information. This makes Elkin and Neimann's algorithm essentially optimal (barring sublinear factors). Crucially, this does not mean that *for every network*, a routing scheme with stretch k necessarily requires each device to store $\Omega(n^{\frac{1}{k}})$ bits of routing information. It simply means that there *exist* pathologic network topologies where this amount of memory is needed.

In practice, most network topologies are restricted to resemble particular graph classes that exclude the worst-case topologies. For example, many network backbone structures resemble trees or series-parallel graphs. Large-scale networks can often be represented by planar graphs where edges never cross. For these restricted topologies, there are more sophisticated algorithms that compute better routing schemes with a stretch of $(1+\epsilon)$ where each device only needs to store $O(\epsilon^{-1} \operatorname{polylog} n)$ bits [Th004, AG06]. However, these algorithms are difficult to translate into a distributed setting as, in many cases, they require complex global calculations that require superlinear time. Again, these calculations require that large parts of the topology are known to single nodes.

Our new approach to overcome this is to avoid these complex calculations altogether. We show that in many practical graph classes, one can circumvent this and compute routing schemes with almost the same qualities in a fully distributed way. Essentially, we reduce the computation of a routing scheme with stretch $(1+\epsilon)$ to $O(\epsilon^{-2} \cdot \log n)$ applications of an $(1+\epsilon)$ approximate shortest path algorithm. On a high level, our idea is to sequentially sample $O(\epsilon^{-2} \cdot \log n)$ $(1+\epsilon)$ approximate shortest paths and then compute routing paths based on the random paths. Note that the number of shortest computations depends on ϵ^{-2}, i.e., the quality of the routing paths. The closer we get to the shortest paths, i.e., the smaller the stretch is, the more computations we need. Further, the hidden constants in the $O(\cdot)$-notation depend on the properties of the input graph. Thus, we show that computing low-stretch routing paths is not significantly harder than computing an approximate SSSP.

Moreover, our techniques for constructing low-stretch routing schemes can be extended to other *distance-based* problems. Notably, they can also be used to compute so-called low-diameter decompositions of the input graphs. Here, we divide the graph into connected subgraphs of bounded diameter. Crucially, in a good decomposition, there are very few edges between these subgraphs. This is a helpful building block of numerous distributed divide-and-conquer algorithms. Further, we present an algorithm to approximate simple optimization problems in which the solution depends on devices at a given distance. With the help of our newly developed techniques, we are able develop clustering algorithms that almost match the best known sequential algorithms. The main takeaway from the second part of this thesis is:

Takeaway 2

(Approximate) SSSP suffices to build low-stretch routing schemes, low-diameter decompositions, and generalized dominating sets for various (practical) network topologies!

As a result, all of our algorithms have essentially the same runtime as approximate shortest-path algorithms. These algorithms are well-understood. In particular, in a recent breakthrough, Rozhon ⓡ[1] al. present an approximate shortest path algorithm that is time-optimal in all relevant models of distributed computation [RGH$^+$22]. This implies that our algorithms are also time-optimal (within polylogarithmic factors). In Part II of this thesis, we provide the exact formal definitions of all these problems. Further, we introduce some technical results, namely, the efficient construction of so-called (weak) separators using only approximate shortest paths. Given an efficient algorithm for computing (weak) separators, we present algorithms for routing schemes, decompositions, covering problems, and their analysis.

1.2 Preliminaries & Notations

Before we start with the main part of the thesis, we present some useful notations and important concepts that we will need to describe our problems, algorithms, and analysis more precisely. First, we will review some basic concepts from graph theory as natural way to model a communication network is with a graph $G = (V, E)$ where the nodes V denote the devices in the edges $E \subseteq V^2$ denote connections. Further, all of our algorithms will make heavy use of randomization. Therefore, we will introduce some basic probability theory, especially tail estimates, that we will use throughout this thesis. Note that we only cover the basic notions required to understand our algorithms and their analysis. For a better reading flow, we will introduce more specific bounds and concepts relevant to only one particular lemma *on-demand* when we require them. Last, we refer to the standard textbooks for a broader and more detailed introduction. In particular, for graph theory, we refer to textbooks of Diestel [Die10] or Mohar [MT01]. For an introduction to probability theory, we recommend the excellent book by Mitzenmacher and Upfal [MU05] and for a wonderful overview on tail estimates, the habilitation by Scheideler [Sch00].

That being said, this section is subdivided into the following four subsections: In Section 1.2.1, we introduce important notions from graph theory and our most important notations. Then, in Section 1.2.2, we introduce several restricted graph classes that can be used to faithfully model modern networks and also have favorable algorithmic properties. Section 1.2.3 introduces the $O(\cdot)$- and $\tilde{O}(\cdot)$-notation that will help us to express the complexity of our algorithms. Finally, Section 1.2.4 gives a primer on probability theory.

1.2.1 Basic Terms from Graph Theory

We begin with some general terms that will be used throughout this thesis. We denote $G = (V, E)$ as a graph with $n := |V|$ nodes and $m := |E|$ edges. G may be directed in which case we denote an edge from v to w as (v, w). Otherwise, if G is undirected, we write $\{v, w\}$. Sometimes, we will also consider bidirected graphs. These are directed graphs, but for every edge $(v, w) \in E$, the *reverse* edge (w, v) is also in E. If G is directed,

[1] The order of authors was randomized. The authors requested that this symbol is used to indicate this.

a node's *outdegree* denotes the number of outgoing edges. Analogously, its *indegree* denotes the number of incoming edges. If G is undirected, then the in- and outdegree are equal. A node's *degree* is the sum of its in- and outdegree. If the graph is undirected or bidirected, we use $\mathsf{deg}(v)$ for the degree of node v. A graph's degree is the maximum degree of any node, which we denote by Δ. We say that a graph is *weakly connected* if there is a (not necessarily directed) path between all pairs of nodes. Sometimes, we will also consider *weighted* graphs that we denote as $G = (V, E, \ell)$ where $\ell : E \to \mathbb{R}$ is a function that assigns a real weight to each edge. For our algorithms, we will typically assume that the weights are non-negative and bound by n^c for some constant $c > 1$. The former is a simplifying assumption that prevents the graphs from having cycles of negative weight. Many of the problems we consider are not well-defined in graphs with such cycles. The latter is a common assumption in the domain of distributed algorithms as we must be able to decode the distances in a finite number of bits to send them to other nodes.

In this thesis, we will often need to argue about the *distance* between two nodes or between sets of nodes. In particular, we must distinguish between the *weighted* and the *hop distance*. We will use $\mathsf{dist}(v, w)$ for the *hop distance*. For a node pair $v, w \in V$, this is the number of nodes in the shortest path between v and w in the *unweighted* version of G. As we will see, this is exactly the number of steps a message sent from nodes v takes to reach node w in the CONGEST or LOCAL model (cf. Section 1.3.1). This notion of distance will be of great importance in the first part of the thesis. Further, we denote the (weighted) distance between a node $v \in V$ and a set $S \subset V$ as $d(v, S)$. This distance is defined as the (weighted) length of the shortest path between v and the node $w \in S$ closest to v. In a weighted graph, it is the minimal sum of weights. If we consider the distance w.r.t. to a subgraph $H \subset G$, we write $d_H(\cdot, \cdot)$. Further for a subset $S \subseteq V$, a subgraph $H \subseteq G$, and a distance δ, we define $B_H(S, \delta) = \{v \in V \mid d_H(v, S) \leq \delta\}$ to be the so-called *ball* that contains all nodes in distance (at most) δ to any node in G. Again, for $H = G$, we omit the subscript. These notions will be of great importance in the second part of the thesis.

A graph's *diameter* is the length of the longest shortest path in G. Based on our two notions of distance, we also distinguish between two *types* of diameter. First, there is the hop diameter HD that denotes the diameter of the *unweighted* version of G. On the other hand, the *weighthed* diameter $\mathcal{D}$ of the graph denotes the length of the weighted longest shortest path. Note that this path could contain significantly less or more than HD nodes. While HD is purely determined by the graph's topology, $\mathcal{D}$ highly depends on the edge weights. If we only consider unweighted graphs, we may use hop diameter and weighted diameter interchangeably as, in this case, they are the same.

In addition to these notions of diameter, we also need measures for the connectivity of a graph. For a subset $S \subseteq V$, we denote $\overline{S} := V \setminus S$. We define the *cut* $c(S, \overline{S})$ as the set of all edges $(v, w) \in V$ with $v \in S$ and $w \in \overline{S}$. We define the number edges that cross a *cut* $c(S, \overline{S})$ as $O_S := |c(S, \overline{S})|$. Our analysis in the first part of this thesis will heavily rely on the conductance of the communication graph. The conductance of set $S \subset V$ is the ratio of its outgoing edges and all its edges. The conductance Φ of a graph G is the minimal conductance of every subset that contains less than $n/2$ nodes. Formally, the conductance is defined as follows:

Definition 1.1 (Conductance). *Let $G := (V, E)$ be a connected Δ-regular graph and $S \subset V$ with $|S| \leq \frac{|V|}{2}$ be any subset of G with at most half its nodes. Then, the conductance $\Phi(S) \in (0, 1)$ of S is*

$$\Phi(S) := \frac{|\{(v, w) \in E \mid v \in S, w \notin S\}|}{\Delta|S|} = \frac{O_S}{\Delta|S|}. \tag{1.1}$$

The conductance $\Phi(G)$ of G is

$$\Phi_\delta(G) := \min_{S \subset V, |S| \leq \frac{|V|}{2}} \Phi(S). \tag{1.2}$$

We further need the notion of small-set conductance, which is a natural generalization of conductance. Instead of denoting the minimum conductance of all sets smaller than $n/2$, small-set conductance only considers sets of size $\frac{\delta|V|}{2}$ for any $\delta \in (0, 1]$. Analogous to the conductance, it is defined as follows:

Definition 1.2 (Small-Set Conductance). *Let $G := (V, E)$ be a connected Δ-regular graph and $S \subset V$ with $|S| \leq \frac{\delta|V|}{2}$ be any subset of G. The small-set conductance Φ_δ of G is*

$$\Phi_\delta(G) := \min_{S \subset V, |S| \leq \frac{\delta|V|}{2}} \Phi(S). \tag{1.3}$$

If it is unclear from context to which graph we are referring, we write $\Phi_G(S)$ and $\Phi_{\delta,G}(S)$ respectively.

Finally, we will, on several occasions, need to measure a graph's *sparsity*. For this, we use the graph's so-called *arboricity*. The arboricity of an undirected graph is the minimum number of forests into which its edges can be partitioned. Clearly, a tree, the sparsest possible connected graph, has arboricity 1. Further, it is easy to verify that a clique, the densest possible connected graph, has an arboricity of n. In general, the arboricity can take any value between the average and the maximum degree.

1.2.2 Restricted Graph Classes

For the second part of the thesis, we require some knowledge of graph topology and certain restricted graph classes. These graph classes exclude pathological worst-case graphs for which all distributed algorithms must have a high runtime or cannot compute a good solution. Still, they are versatile enough to model many real-world networks. In the following, we give a short overview that is sufficient to verify our main claims. For more details on graph theory and restricted graphs classes, we refer to textbooks such as [Die10] or [MT01].

Graphs of Bounded Treewidth We begin our short tour with graphs of bounded treewidth. Informally speaking, the treewidth quantifies how far a given graph is from being a tree. The smallest treewidth is 1; the graphs with treewidth 1 are precisely the trees and the forests. For a formal definition, we require the concept of tree decompositions, which are defined as follows:

Definition 1.3 (Tree Decomposition). *A tree decomposition of a graph $G = (V, E)$ is a tree T with nodes $X_1, \dots, X_n$, with $X_i \subset V$ such that the following three properties hold:*

1. *The union of all sets X_i equals V.*

2. *If X_i and X_j both contain v, then all nodes X_k in the (unique) path between X_i and X_j in T contain v as well.*

3. *For every edge $(v, w) \in E$ in the graph, there is a subset X_i that contains both v and w.*

The width of a tree decomposition is $\max_{X_i \in T} |X_i| - 1$. The treewidth $\tau(G)$ of a graph G is the minimum width among all possible tree decompositions of G.

Planar Graphs & Graphs of Bounded Euler Genus Moving on to the next class, a planar graph $G = (V, E)$ is a graph that can be drawn into the Euclidean plane $\mathbb{R}^2$ without two edges crossing. By *drawing*, we mean that nodes are mapped to points and edges to curves/lines between these points. A drawing of a planar graph is called a *geometric planar embedding*. More generally, if the graph can be drawn on a surface with genus g, i.e., a surface with g holes[2], we say the graph has Euler-genus g. Note that the Euclidean plane has a genus of 0, so planar graphs have an Euler-genus 0.

Graphs Excluding a Fixed Minor Finally, we move to graphs that exclude a fixed minor. An undirected graph H is called a *minor* of the graph G if H can be obtained from G by (repeatedly) using any of the following operations: Deleting edges, deleting nodes, and contracting edges. While the former two are self-explanatory, the latter is more subtle. If we contract an edge $\{v, w\}$, we merge its two endpoints into a new supernode that is adjacent to all neighbors of v and w. Interestingly, many graph classes can be characterized by their *forbidden* minors. A graph class $\mathcal{G}$ has a forbidden minor F if, for every $G \in \mathcal{G}$, it holds that F is not a minor of G. In other words, obtaining F from G is impossible by deleting edges, deleting nodes, and contracting edges.

Trees are an easy example of such a graph class. By definition, a tree does not contain a cycle, and the three permitted operations cannot create a cycle as they only remove edges. Therefore, no minor of a tree can contain a cycle and, thus, is still a tree. The smallest graph that contains a cycle is K_3, the clique of 3 nodes. Following our argument, no tree can contain K_3 as minor. On the other hand, any graph G that is not a tree *must* contain K_3: Since G is not a tree, it must have a cycle $C = (v_1, v_2, v_3, \ldots, v_l, v_1)$. We delete all nodes not on this cycle and iteratively contract all edges between v_3 and v_l. Thus, we end up with K_3. Of course, the theory of graph minors offers more than this pet example. Famously, Wagner's theorem states that a graph is planar if and only if its minors include neither the K_5 nor the complete bipartite graph $K_{3,3}$ [Wag37]. Note that these graph classes have received significant attention in the context of distributed algorithms (cf. [GH16a, GH16b, GH21, GP17, IKNS22]). Not only do they contain graph topologies that frequently appear in practice, but they also have favorable properties for distributed algorithms, as they exclude pathologic worst-case topologies.

Universally k-path Separable Graphs Finally, we introduce universally k-path separable graphs, a helpful graph class with many relevant graph classes and useful properties for our algorithms. In contrast to the previously mentioned graph classes, this class is relatively new. It was introduced by Abraham and Gavoille in 2006 [AG06]. As the name suggests, we require the notion of separators to define these graphs. For a given

[2]The exact definition of a hole in the context of topology is beyond the scope of the thesis and also necessary to understand our algorithmic contributions. We refer to [MT01] or various online resources for more details

graph $G = (V, E)$, a separator S is a *subgraph* of G such that in the induced subgraph $G \setminus S$ that results from removing S from G, every connected component only contains a constant fraction of G's nodes.

In many applications, one is interested in finding *small* separators that consist of as few nodes as possible. For example, it is long known that any planar graph contains a separator that consists of at most $O(\sqrt{n})$ nodes [LT80]. The result can be generalized to the larger class of K_r-free graphs [KL24]. However, for the problems of clustering and routing that we will consider in Part II, one is not interested in such small separators in the classical sense. Instead, we want to consider separators that consist of a few short(est) paths. This means that the separators could perhaps contain a linear fraction of the nodes, but all these nodes lie on short paths. As we will see, the structural properties of the separators have several useful properties for the problems we are tackling. More precisely, we will consider the so-called k-path separable graphs that Abraham and Gavoille introduced in [AG06]. Roughly speaking, a weighted graph $G = (V, E, \ell)$ is k-path separable if the *recursive* removal of k shortest paths results in connected components containing a constant fraction of the nodes. Moreover, a graph $G = (V, E)$ is *universally* k-path separable if it is k-path separable for *every* weight function ℓ. Note that this makes universal k-path separability a topological property rather than a property that depends on the weight function. Abraham and Gavoille formalized this generic class of path separators as follows:

Definition 1.4 (Universally k-path Separable Graphs [AG06, DG10]). *A weighted graph $G := (V, E, \ell)$ with n vertices is k-*path separable*, if there exists a subset of vertices S, called a k-path separator, such that:*

1. *$S := \mathcal{P}_0 \cup \mathcal{P}_1 \cup \mathcal{P}_2 \cup \ldots$, where each $\mathcal{P}_i := \{P_{i_1}, P_{i_2}, \ldots\}$ is a set of shortest paths in $G \setminus \bigcup_{j<i} \mathcal{P}_j$.*
2. *The total number of paths in S is at most k, i.e., $\sum_{\mathcal{P}_i \in S} |\mathcal{P}_i| \leq k$.*
3. *Each connected component in $G \setminus S$ contains at most $\frac{n}{2}$ nodes.*
4. *Either $G \setminus S$ is empty or k-path separable.*

If G is k-path separable for any weight function ℓ, we call G universally k-path separable. We will sometimes abuse notation and use $P_{i_j} \in S$ when we refer to some path $P_{i_j} \in \mathcal{P}_i$ and $\mathcal{P}_i \in S$.

Given their definition, a natural question is which graph classes are k-path separable. Clearly, all graphs that have separators that consist of at most η vertices are trivially universally η-path separable. This follows because every node can be seen as the shortest path to itself in any subgraph it is contained in. Notably, graphs of bounded tree-width τ have separators that consist of τ nodes [RS86] and are therefore universally τ-path separable as under any cost function on the edges, nodes are shortest paths to themselves. It holds:

Lemma 1.1 (Theorem 1 in [DG10]). *A graph of treewidth τ is universally τ-path separable.*

Further, due to Thorup, it is known that planar graphs are universally 3-path separable [Tho04]. It holds:

Lemma 1.2 (Lemma 2.3 in [Tho04]). *A planar graph is universally 3-path separable.*

The proof in [Tho04] is based on the so-called fundamental-cycle method, which in every spanning tree finds two tree paths whose removal disconnects the graph into smaller connected components. Similarly, if G has an Euler-genus g, i.e., it can be embedded into a surface with genus g the following holds: There exists a cycle A comprised of two shortest paths emanating at a common root, such that $G \setminus A$ has Euler-genus at most $g-1$. This has been claimed in [AGG$^+$19a] based on a proof presented in [MT01]. Thus, by iteratively removing these cycles, we eventually end up in a graph of genus 0, i.e., a planar graph that is 3-separable Thus, it holds:

Lemma 1.3 (Corollary of Lemma 21 in [AGG$^+$19a]). *A graph of Euler-genus g is universally $O(g)$-path separable.*

Finally, Abraham and Gavoille showed that every graph $G := (V, E, \ell)$ that does not include a fixed clique minor K_r is universally k-path separable where $k := f(r)$ depends only on r and not the size of G [AG06]. It holds:

Lemma 1.4 (Theorem 1 in [AG06]). *If G does not contain K_r as a minor, then G is universally $f(r)$-path separable. Here, $f(r)$ is a function that only depends on r and not on n*

We note that $f(r)$, albeit being a constant, is extremely large. Its concrete value is based on the Structure Theorem of Seymour and Robinson [RS03]. Their analysis is quite involved and stretches over a vast series of papers and articles that lead to their result in [RS03]. Moreover, due to its complexity, it is hard even to find an asymptotic characterization of that number. In [LSZ20], it is described as a "tower of powers of 2 of height at most 5, with r^{1000} on top". We emphasize that this number very likely is too high for practical applications. Nevertheless, it offers us theoretical insights.

Summing up, k-path separable graphs are a superclass of a large set of restricted graph classes. Thus, if we develop an algorithm for universally k-path separable graphs whose runtime and output depend on k, we also obtain novel results for planar graphs, graphs of bounded treewidth, and, most importantly, K_r-free graphs. However, the dependence k might not be optimal for a specific class.

1.2.3 Basic Terms from Complexity Theory

For the most part, we express the complexity of our algorithms in the well-known $O(\cdot)$-notation. However, especially in the second part of this thesis, we will make heavy use of the $\tilde{O}(\cdot)$-notation. Similar to the $O(\cdot)$-notation that suppresses constant factors, the $\tilde{O}(\cdot)$-notation suppresses polylogarithmic factor. To be precise, in the $O(\cdot)$-notation, it holds:

$$\begin{aligned}
f(n) \in O(g(n)) &\Leftrightarrow \exists n_0 > 0 : \exists c > 0 : \forall n \geq n_0 : f(n) \leq c \cdot g(n)\\
f(n) \in \Omega(g(n)) &\Leftrightarrow \exists n_0 > 0 : \exists c > 0 : \forall n \geq n_0 : f(n) \geq c \cdot g(n)\\
f(n) \in \Theta(g(n)) &\Leftrightarrow (f(n) \in O(g(n))) \wedge (f(n) \in \Omega(g(n)))
\end{aligned}$$

Whereas in the $\tilde{O}(\cdot)$-notation, it holds:

$$\begin{aligned}
f(n) \in \tilde{O}(g(n)) &\Leftrightarrow \exists n_0 > 0 : \exists c > 0 : \forall n \geq n_0 : f(n) \leq \log^c n \cdot g(n)\\
f(n) \in \tilde{\Omega}(g(n)) &\Leftrightarrow \exists n_0 > 0 : \exists c > 0 : \forall n \geq n_0 : f(n) \geq g(n)/\log^c n\\
f(n) \in \tilde{\Theta}(g(n)) &\Leftrightarrow (f(n) \in \tilde{O}(g(n))) \wedge (f(n) \in \tilde{\Omega}(g(n)))
\end{aligned}$$

This notation is helpful when the $g(n)$ is (asymptotically) much larger than $\log^c n$ for any constant $c > 0$. For example when $g(n) \in \Omega(\sqrt{n})$. Further, many complex graph algorithms have runtime that depends on some property P of the input graph. However, they often rely on using other algorithms in a black-box, and therefore, polylogarithmic factors quickly stack up and become increasingly hard to trace. Thus, it has become more or less standard to use the $\tilde{O}(\cdot)$-notation, which emphasizes the superlogarithmic factors of runtime, i.e., to write $\tilde{O}(P)$ instead of $O(P \cdot \log^c n)$ as it emphasizes the dependency on P, which is more interesting from a theoretical standpoint. We will also follow this established convention. Of course, we acknowledge that the hidden polylogarithmic factors make a significant difference in practice. Much more so than the hidden constants in the $O(\cdot)$-notation. However, we emphasize again that our goals are theoretical in nature.

1.2.4 Basic Terms from Probability Theory

Finally, this section establishes well-known bounds and tail estimates from probability theory that will be used in all our results. First and foremost, we note that almost all of our algorithms are randomized. Therefore, they have a small probability of *failure*, .e.g., through taking a longer time, sending more messages, or producing a false solution altogether. This raises the obvious question of what we consider a *small* or, conversely, a *high* probability. To this end, recall that our algorithms are executed on a network with n nodes. We use the *de-facto* standard measure for randomized algorithms and say that an event holds *with high probability* if it holds with probability at least $1 - {}^{1}/{n^c}$ for some $c \geq 1$. In other words, the probability of failure is polynomially small in n. That means the larger the networks get, the less likely it is to fail. Therefore, the success probability scales with the size of the network. The concrete value of c is a so-called *tunable* constant. Typically, it depends on the algorithm's message complexity and/or runtime, such that a larger message or time complexity can be traded with a higher probability of success. In all of our algorithms, this relationship is exponential, so the constant c can be hidden in the $O(\cdot)$-Notation for a succinct notation. For example, if we write something like "*the algorithm requires $O(T)$ time steps, w.h.p.*", we mean that "*there is a constant $c \geq 1$ such that algorithm requires $O(c \cdot T)$ time steps with probability $1 - {}^{1}/{n^c}$*."

Having introduced this central concept, let's continue with some useful tail estimates that we will use throughout this thesis. The first bound is Markov's inequality, which estimates the probability of a random variable reaching a certain value based on its expectation. It holds:

Lemma 1.5 (Markov's Inequality). *Let X be a non-negative random variable and $a > 0$, then it holds:*

$$\mathbf{Pr}[X \geq a] \leq \frac{\mathbb{E}[X]}{a} \tag{1.4}$$

While this inequality applies to various variables, it is not very precise. For more precise bounds, we heavily use the well-known Chernoff bound, another standard tool for analyzing distributed algorithms. In particular, we will use the following version:

Lemma 1.6 (Chernoff Bound). *Let $X = \sum_{i=1}^{n} X_i$ for independent distributed random variables $X_i \in \{0,1\}$ and $\mathbb{E}(X) \leq \mu_H$ and $\delta \geq 1$.*

$$\mathbf{Pr}[X > (1+\delta)\mu_H] \leq e^{-\left(\frac{\delta \mu_H}{3}\right)}, \tag{1.5}$$

Similarly, for $\mathbb{E}(X) \geq \mu_L$ *and* $0 \leq \delta \leq 1$ *we have*

$$\mathbf{Pr}[X < (1-\delta)\mu_L] \leq e^{-\left(\frac{\delta^2 \mu_L}{2}\right)} \tag{1.6}$$

Further, we use the union bound that helps us to bound the probability for many correlated events as long as all these events have a very small probability of happening. It holds:

Lemma 1.7 (Union Bound). *Let* $\mathcal{B} := B_1, \ldots, B_m$ *be a set of* m *(possibly dependent) events. Then, the probability any of the events in* $\mathcal{B}$ *happens can be bounded as follows:*

$$\mathbf{Pr}\left[\bigcup_{i=1}^{m} B_i\right] \leq \sum_{i=1}^{m} \mathbf{Pr}[B_i] \tag{1.7}$$

This bound is tremendously helpful when dealing with a polynomial number of *bad* events, say n^c many, that do not happen with high probability, say $1 - n^{c'}$ for some tunable constant c'. If we choose this constant c' big enough, the union bound trivially implies that the probability of any bad event happening is $n^{-c''}$ for a constant $c'' := c' - c$. Thus, if we can show that a specific event holds for a single node w.h.p., the union bound implies that it holds for all nodes w.h.p. Due to this fact, we will often implicitly apply the union bound when we consider a polynomial number of events.

1.3 Model(s)

This work presents and rigorously analyzes algorithms for modern communications networks like the Internet. Therefore, it is of utmost importance to choose a suitable computational model that represents the properties of such a network. This, of course, begs the question of precisely what these properties are. As we have established in the introduction, there are many aspects to communication networks depending on the use case. Let's consider an algorithm that computes routes in the static backbone network to find the best paths between data centers with dedicated physical connections to each other. This setup vastly differs from an algorithm in a peer-to-peer network executed by possibly volatile end-user devices. Therefore, a *catch'em-them-all* model that combines all *layers* of the network is not the right choice. In this section, we will give an overview of the different models used in this thesis that are tailored to our specific problem setups.

1.3.1 The CONGEST and LOCAL model

In the classical CONGEST and LOCAL model[Pel00], we consider a static graph $G = (V, E)$ that consists of n nodes and m edges. Both these models aim to faithfully represent classical cable networks that, for example, form the Internet's *backbone*. Each node represents a stationary device like a server or data center. Each edge represents a physical (cable) connection between two stationary devices. In both models, each node has a unique identifier. We assume these identifiers consist of $O(\log n)$ bits, i.e., they are picked from some interval $[1, \ldots, n^c]$ for some $c \in \Theta(1)$. Time proceeds in synchronous rounds. In each round, nodes receive *all* messages sent in the previous round; they perform a finite computation based on their internal states and the messages they have received so far; Finally, they send distinct messages to their neighbors in G. Thus, a node of degree $\mathtt{deg}(v)$ receives up to $\mathtt{deg}(v)$ distinct messages in the next time step. The only (yet *very* important) difference between CONGEST and LOCAL lies in the size of these messages. In LOCAL, the size of the messages is *unbounded*. This allows for the design of algorithms in which all nodes continuously relay all messages they have received thus far to their neighbors. That means, in ℓ rounds of communication, a node may learn the complete state of its ℓ-neighborhood. Therefore, LOCAL is a great model to prove lower bounds as it can answer the question of what information is theoretically available to a node after ℓ rounds. However, this also means that each node can learn the whole of topology of G and all nodes' inputs with HD rounds. This implies that every problem in LOCAL can be solved in at most HD time by first aggregating the topology and then letting each node solve the problem locally. For large networks like the Internet, this approach does not scale as it leads to large messages containing $O(m \log n)$ bits, i.e., the complete topology. In CONGEST, a node $v \in V$ may send a *distinct* message of size $O(\log n)$ to each neighbor in G[3]. Thus, a message may contain a node's identifier and a few more bits of information.

1.3.2 The NCC_0 and P2P-CONGEST model

Next, we present models for overlay networks. Recall that overlays are *virtual* networks built on top of other networks, the so-called *underlay*. The underlay allows for point-point communication between two devices based on identifiers. In particular, if the identifier of a device is known, the underlay allows anyone to send a message to this device. This abstracts away the specifics and minutiae of the underlying network and reduces it to a pure communication interface. The internet is classical for this type of network. Given the IP address of a device, *any* device connected to the internet can (in principle[4]) send a message to it.

While many theoretical works consider algorithms for overlay networks, to the best of our knowledge, there is no *unified* model with the same widespread adoption as CONGEST or LOCAL on *classical* distributed computing. However, nearly all models share some common concepts. Representative of these models, in this work, we use the so-called NCC_0 model [ACC$^+$20], which is a variant of the general *Node-Capacitated Clique* (NCC) model for overlay networks [AGG$^+$19b], and the P2P-CONGEST model [GPRT20]. We have a fixed node set V with n nodes in both models. Each node $u \in V$ has a unique *identifier* $\text{id}(u)$, which is a bit string of length $O(\log n)$. Further, time proceeds in *synchronous rounds*. We represent the network as a *directed* graph $G = (V, E)$, where there is a directed edge $(u, v) \in E$ if u knows $\text{id}(v)$, i.e., it holds:

$$E := \{(u, v) \mid u \text{ knows } \text{id}(v)\}$$

Initially, all nodes are connected in a weakly connected graph G_0. If u knows $\text{id}(v)$ in round i, then it can send a message to v that will be received at the beginning of round $i + 1$. New connections can be established by sending node identifiers: if u sends $\text{id}(w)$ to v, then v can establish an edge (v, w). We restrict the size of a message to $O(\log n)$ bits, which allows a message to carry a constant number of identifiers.

Again, the crux of the different models is the number of messages each node can send in each round. As each node represents a simple device connected to the Internet, we cannot expect to be able to send a massive

[3] Some works relax this to $\tilde{O}(1)$ bits, which is still a far cry from $O(m \log n)$.

[4] Of course, in practice, one needs to account for NATs and Firewalls, but we do not consider this due to the theoretical nature of this work.

amount of data in a single step. Thus, we need to limit the total number of messages each node can *send* and *receive* in each round to obtain a faithful model.

In the NCC_0 model, if a node receives more than $O(\log n)$ messages, an arbitrary subset of these messages is dropped. Our algorithms will *not* exploit this and ensure that $O(\log n)$ messages are received. The bound of $O(\log n)$ is arguably a natural choice, preventing algorithms from being needlessly complicated while ensuring scalability. In the P2P-CONGEST model, each node $v \in V$ can send $O(\mathtt{deg}(v) \log n)$ messages where $d(v)$ is its degree in the **initial** overlay G_0. For high degrees, this, to a certain extent, juxtaposes the idea that a node has limited sending capabilities. However, from a theoretical point of view, this allows us to more easily design algorithms that work on *any* initial topology without considering the initial degrees and finding *workarounds* for high degree nodes. This makes it easier to explain the core idea behind some algorithms. Further, note that for some problems, a node *must* send or receive a message to or from all its neighbors in G_0. If the initial graph has a constant degree, both models are equivalent.

1.3.3 The HYBRID model

Finally, we note that there are modern networks that may not strictly fall into either of the categories sketched above. Consider, for example, so-called wireless sensor networks. These are networks of minimalistic devices, so-called sensor nodes, equipped with sensors that monitor their environment. The sensors record conditions such as temperature, sound, or pressure. Further, each device can communicate with nearby devices via wireless communication. This allows them to aggregate the recorded data and perform various other tasks to maintain the network without central control. This network could be modeled as a graph $G := (V, E)$, where each node $v \in V$ represents a device and each edge $\{v, w\} \in E$ signals that the two endpoints are in each other's communication range. We can then use CONGEST or LOCAL to model the communication in this graph. However, given the ubiquitous availability of an internet connection through 5G and satellite uplinks, a networking device may have access to the internet via a separate antenna. Therefore, they can also send messages to other devices using the internet. Given that some IP addresses of other devices are known, this type of communication can be represented through the NCC_0 model. Thus, the communication in these networks can be modeled as a mix of the previously mentioned models. The list of distributed systems that leverage multiple communication modes does not end here. For instance, hybrid data center networks employ both high-speed optical or wireless circuit switching technologies and traditional electronic packet switches to improve throughput [FPR+10, HKP+11]. Organizations can also use dynamic multipoint VPNs or so-called hybrid WANs to supplement their internal network with connections via the Internet [TBKC18].

The HYBRID model was introduced in [AHK+20a] as a means to study such distributed systems that leverage multiple communication modes of different characteristics. In particular, the HYBRID model considers networks with so-called *local* and a *global* modes. In the *local* communication mode, nodes have a large bandwidth. However, communication is restricted to edges of a (static) *graph* G. Further, they possess a *global* communication mode where, in principle, any two nodes may communicate. Still, very little such communication can take place per unit of time. More precisely, the *local* communication mode is modeled as a connected graph, in which each node is initially aware of its neighbors and is allowed to send a message of size λ bits to each neighbor in each round. In the *global* communication mode, each round, each node may send or receive

γ bits to/from every other node. An arbitrary subset of messages is dropped if any restrictions are violated in a given round. Again, our algorithms will never exploit this and ensure that no messages are dropped.

In this thesis, we consider a weak form of the HYBRID model. In the first part where we want to optimize runtimes down to logarithmic factor, we set $\lambda \in O(\log n)$ and $\gamma \in O(\log^3 n)$. In the second part where the logarithmic factors in the runtime are less important, we set $\lambda \in O(\log n)$ and $\gamma \in O(\log^2 n)$, which corresponds to the combination of the classic distributed models CONGEST as local mode, and NCC_0 presented as global mode. The latter is the typical setup for the global mode that is used in most works on the topic. Note that some previous papers that consider hybrid models use $\lambda = \infty$, i.e., the LOCAL model as local mode.

1.4 List of Own Publications

1. Thorsten Götte, Christian Scheideler, and Alexander Setzer. On underlay-aware self-stabilizing overlay networks. In Taisuke Izumi and Petr Kuznetsov, editors, *Stabilization, Safety, and Security of Distributed Systems - 20th International Symposium, SSS 2018, Tokyo, Japan, November 4-7, 2018, Proceedings*, volume 11201 of *Lecture Notes in Computer Science*, pages 50–64. Springer, 2018

2. Thorsten Götte, Vipin Ravindran Vijayalakshmi, and Christian Scheideler. Always be two steps ahead of your enemy. In *2019 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2019, Rio de Janeiro, Brazil, May 20-24, 2019*, pages 1073–1082. IEEE, 2019

3. Thorsten Götte, Kristian Hinnenthal, and Christian Scheideler. Faster construction of overlay networks. In Keren Censor-Hillel and Michele Flammini, editors, *Structural Information and Communication Complexity - 26th International Colloquium, SIROCCO 2019, L'Aquila, Italy, July 1-4, 2019, Proceedings*, volume 11639 of *Lecture Notes in Computer Science*, pages 262–276. Springer, 2019

4. Michael Feldmann, Thorsten Götte, and Christian Scheideler. A loosely self-stabilizing protocol for randomized congestion control with logarithmic memory. In Mohsen Ghaffari, Mikhail Nesterenko, Sébastien Tixeuil, Sara Tucci, and Yukiko Yamauchi, editors, *Stabilization, Safety, and Security of Distributed Systems - 21st International Symposium, SSS 2019, Pisa, Italy, October 22-25, 2019, Proceedings*, volume 11914 of *Lecture Notes in Computer Science*, pages 149–164. Springer, 2019

5. Thorsten Götte, Kristian Hinnenthal, Christian Scheideler, and Julian Werthmann. Time-optimal construction of overlay networks. In Avery Miller, Keren Censor-Hillel, and Janne H. Korhonen, editors, *PODC '21: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, July 26-30, 2021*, pages 457–468. ACM, 2021

6. Thorsten Götte, Christina Kolb, Christian Scheideler, and Julian Werthmann. Beep-and-sleep: Message and energy efficient set cover. In Leszek Gasieniec, Ralf Klasing, and Tomasz Radzik, editors, *Algorithms for Sensor Systems - 17th International Symposium on Algorithms and Experiments for Wireless Sensor Networks, ALGOSENSORS 2021, Lisbon, Portugal, September 9-10, 2021, Proceedings*, volume 12961 of *Lecture Notes in Computer Science*, pages 94–110. Springer, 2021

7. Thorsten Götte and Christian Scheideler. Brief announcement: The (limited) power of multiple identities: Asynchronous byzantine reliable broadcast with improved resilience through collusion. In Kunal Agrawal and I-Ting Angelina Lee, editors, *SPAA '22: 34th ACM Symposium on Parallelism in Algorithms and Architectures, Philadelphia, PA, USA, July 11 - 14, 2022*, pages 99–101. ACM, 2022

8. Thorsten Götte, Kristian Hinnenthal, Christian Scheideler, and Julian Werthmann. Time-optimal construction of overlay networks. *Distributed Comput.*, 36(3):313–347, 2023 (**Special Issue of** PODC'21)

9. Thorsten Götte, Christina Kolb, Christian Scheideler, and Julian Werthmann. Beep-and-sleep: Message and energy efficient set cover. *Theor. Comput. Sci.*, 950:113756, 2023 (**Special Issue of** ALGOSENSORS'21)

10. Jinfeng Dou, Thorsten Götte, Henning Hillebrandt, Christian Scheideler, and Julian Werthmann. Brief announcement: Distributed construction of near-optimal compact routing schemes for planar graphs. In Rotem Oshman, Alexandre Nolin, Magnús M. Halldórsson, and Alkida Balliu, editors, *Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing, PODC 2023, Orlando, FL, USA, June 19-23, 2023*, pages 67–70. ACM, 2023

11. Jinfeng Dou, Thorsten Götte, Henning Hillebrandt, Christian Scheideler, and Julian Werthmann. Distributed and parallel low-diameter decompositions for arbitrary and restricted graphs. In Raghu Meka, editor, *16th Innovations in Theoretical Computer Science Conference (ITCS 2025), New York City, NY, USA, January 7–10, 2025*, Leibniz International Proceedings in Informatics (LIPIcs), page (to appear), Dagstuhl, Germany, 2025. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik

Part I

Fast Overlay Construction and its Applications

Overview of Part I

In this part of the thesis, we consider algorithms for overlay networks in the NCC_0, P2P-CONGEST, and HYBRID models. The main result presented in this part is an algorithm for the fast construction of overlays from any initial configuration. This algorithm was previously published in the following conference article:

> Thorsten Götte, Kristian Hinnenthal, Christian Scheideler, and Julian Werthmann. Time-optimal construction of overlay networks. In Avery Miller, Keren Censor-Hillel, and Janne H. Korhonen, editors, *PODC '21: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, July 26-30, 2021*, pages 457–468. ACM, 2021

Furthermore, we present several application of this algorithm which use the algorithm to solve a variety of other problems. Most of the additional results were part of the corresponding journal publication of the conference paper, namely

> Thorsten Götte, Kristian Hinnenthal, Christian Scheideler, and Julian Werthmann. Time-optimal construction of overlay networks. *Distributed Comput.*, 36(3):313–347, 2023

The remainder of this part is structured as follows. This introductory chapter gives an overview and a discussion of all results that we present in this part. The main result is the construction of an overlay of logarithmic degree and diameter in $O(\log n)$ time. The corresponding algorithm is presented in Chapter 2. For an easier presentation, the algorithm is stated in the P2P-CONGEST model where a node's communication capabilities scale with its initial degree. In Chapter 3 we extend to algorithm to the HYBRID model where the communication is more limited. We then present three applications in the HYBRID model: For an (undirected) graph G with arbitrary degree, we show how to compute connected components (cf. Chapter 3), a spanning forest (cf. Chapter 4), and a maximal independent set (cf. Chapter 5). For each result, we present a short summary of the main algorithmic ideas and the main challenges of the analysis. We conclude the part by presenting some related work and open issues in Chapter 6.

Note that the journal article [GHSW23] contained a fourth additional result on identifying biconnected components. However, Julian Werthmann conceived and wrote the algorithm in its entirety as part of a seminar (supervised by Kristian Hinnenthal). Therefore, it is not part of this thesis. As for the other results, the author was involved in designing the algorithms, conceiving the analysis, and writing the journal article that is the basis for this chapter.

It is either easy or impossible.

Salvador Dali

2 Time-Optimal Construction Of Overlays

Our first contribution is constructing an overlay network of diameter $O(\log n)$ in $O(\log n)$ rounds in the P2P-CONGEST model starting from an arbitrary initial state. If the initial overlay's degree is constant, the algorithm also works in the NCC_0 model. We present a generalization to the HYBRID model in the next chapter. Note that $O(\log n)$ rounds is the worst-case lower bound for the problem: If the nodes initially form a line, it takes $O(\log n)$ rounds to reduce the diameter to $O(\log n)$, even if every node could introduce all of its neighbors to each other in each round. To see this, consider the two endpoints of the line. Initially, their distance is n and in every step it reduces by at most a constant factor, even if the nodes had umlimited communication capacities.

The concrete topology we are constructing is a so-called *well-formed tree*, which is a rooted tree of degree $O(\log n)$ and diameter $O(\log n)$ that contains all nodes of G. We chose this structure because any *well-behaved* overlay of logarithmic degree and diameter (e.g., butterfly networks, path graphs, sorted rings, trees, regular expanders, De Bruijn graphs, etc.) can be constructed in $O(\log n)$ rounds, w.h.p., starting from a well-formed tree. We present an algorithm that constructs such a tree in $O(\log n)$ time, w.h.p., in the P2P-CONGEST model. In addition to their initial neighborhood, we only assume that all nodes know an approximation of $\log n$, i.e., a very loose polynomial upper bound on n, the number of nodes. This is a mild assumption. Alternatively, the algorithm can be started with geometrically increasing guesses for $\log n$.

Before the publication of this result, the best known algorithm took $O(\log^{3/2} n)$ time, w.h.p. [GHS19b]. However, to the best of our knowledge, there was no $O(\log n)$-time algorithm that can construct a well-defined overlay with logarithmic communication for any topology. Our result finally closed the gap and presented the first algorithm that achieves these bounds, w.h.p. It does so by using a radically different approach, arguably simpler than existing solutions. All of the previous algorithms (i.e., [AAC+05, AW07, GHSS17, GHS19b, GPRT20]) essentially employ the same high-level approach to alternatingly group and merge so-called *supernodes* (i.e., sets of nodes that act in coordination) until only a single supernode remains. However, these supernodes need to be consolidated after being grouped with adjacent supernodes to distinguish internal from external edges. This consolidation step makes it difficult to improve the runtime further using this approach. Thus, instead of arranging the nodes into supernodes (and paying a price of complexity and runtime for their maintenance), we establish random connections between the nodes by performing short *constant length* random walks. Each node starts a small number of short random walks, connects itself with the respective endpoints, and drops all other connections. Then, it repeats the procedure on the newly obtained graph.

The approach is based on classical overlay maintenance algorithms for *unstructured* networks such as, for example, [LS03] or [GMS04] as well as practical libraries for overlays like JXTA [OG02] where the connection are established via random walks. Note that our analysis significantly differs from [LS03] and [GMS04] as we do not assume that nodes arrive one after the other. Instead, we assume an arbitrary initial graph with possibly small conductance. Using novel techniques by Kwok and Lau [KL14] combined with elementary probabilistic arguments, we show that short random walks incrementally reduce the conductance of the graph. Once the conductance is constant, the graph's diameter must be $O(\log n)$. Note that such a graph can easily be transformed into many other overlay networks, such as a sorted ring, e.g., by performing a BFS and applying the algorithm of Aspnes and Wu [AW07] to the BFS tree *or* by using the techniques by Gmyr et al. [GHSS17] Altogether, we show the following theorem:

Theorem 1. *Time-Optimal Construction of Overlays (P2P-CONGEST)*
Let $G = (V, E)$ be a weakly connected directed graph and suppose each node knows values $L, \Lambda \in \Theta(\log n)$. Then, there is a randomized algorithm in the P2P-CONGEST model that constructs a well-formed tree in $O(L)$ rounds, w.h.p. Each node sends at most $O(\mathtt{deg}(v)\Lambda)$ messages per round, w.h.p.

Note that we *could* use techniques from [GHSS17] or [AW07] to further refine our constructions and to reduce the degree of the well-formed tree to $O(1)$. However, since the main focus of our algorithm is the novel idea of using random walks, we omit further details here and refer the reader to [GHSS17] and [AW07]. The remainder of this chapter the structured as follows. In Section 2.1, we describe our main algorithm. In Section 2.2 we establish some basic combinatorial lemmas that will be central to our analyis, which we present in Section 2.3.

2.1 The Overlay Construction Algorithm

In this section, we present our algorithm to construct a well-formed tree in $O(\log n)$ time, w.h.p., and give the proof to establish the correctness of Theorem 1. To the best of our knowledge, our approach is different from *all* previous algorithms for our problem [AS03, AW07, GHSS17, GHS19b] in that it does *not* use any form of clustering to contract large portions of the graph into supernodes. On a high level, our algorithm progresses through $O(\log n)$ iterations, where the next graph is obtained by establishing random edges on the current graph. More precisely, each node of a graph simply starts a few random walks of constant length and connects itself with the respective endpoints. The next graph only contains the newly established edges. We will show that after $O(\log n)$ iterations of this simple procedure, we reach a graph with diameter $O(\log n)$. One can easily verify that this strategy does not trivially work on any graph, as the graph's degree distributions and other properties significantly impact the distribution of random walks. However, as it turns out, we only need to ensure that the initial graph has some *nice* properties to obtain well-behaved random walks. More precisely, throughout our algorithm, we maintain that the graph is *benign*, which we define as follows.

Definition 2.1 (Benign Graphs). *Let $G := (V, E)$ be a simple undirected graph and $\Lambda \in \Omega(\log n)$ and $\Delta \geq 64\Lambda$ be two values (with big enough constants hidden by the Ω-Notation). Then, we call G benign if and only if it has the following three properties:*

1. *(G **is Δ-regular)** Every node $v \in V$ has exactly Δ in- and outgoing edges (which may be self-loops).*
2. *(G **is lazy)** Every node $v \in V$ has at least $1/2\Delta$ self-loops.*
3. *(G **has a Λ-sized minimum cut)** Every cut $c(S, \overline{S})$ with $S \subset V$ has at least Λ edges.*

The properties of benign graphs are carefully chosen to be as weak as possible while still ensuring the correct execution of our algorithm. A degree of $\Delta \in \Omega(\log n)$ is necessary to keep the graph connected. If we only had a constant degree, a standard result from random graphs implies that, w.h.p., there would be nodes disconnected from the graph when sampling new neighbors. If the graphs were not lazy, many theorems from the analysis of Markov chains would not hold as the graph could be bipartite, which would greatly complicate the analysis. This assumption only slows down random walks by a factor of 2. Lastly, the Λ-sized cut ensures that the graph becomes more densely connected in each iteration, w.h.p. In fact, with constant-sized cuts, we cannot easily ensure this property when using random walks of constant length.

We will now describe the algorithm in more detail. Recall that throughout this section, we will assume the P2P-CONGEST model. Further, we assume that the initial graph G has maximum degree Δ_G and is connected. Given these preconditions, the algorithm has four input parameters ℓ, Δ, Λ, and L known to all nodes. Recall that $L \in \Theta(\log n)$ is an upper bound on $\log n$ and determines the runtime. The value $\ell \in \Omega(1)$ denotes the length of the random walks, $\Delta \in O(\log n)$ is the desired degree, and $\Lambda \in O(\log n)$ denotes the (approximate) size of the minimum cut. All of these parameters are *tunable* and the hidden constants need to be chosen big enough for the algorithm to succeed, w.h.p. In particular, the value of Λ will determine the success probability. We discuss this in more detail in the analysis. Finally, we emphasize that Δ_G is the degree of the initial graph G and $\Delta \in O(\log n)$ is the degree of the graph created by the algorithm.

Our algorithm consists of three stages. In the first stage, we ensure that the initial graph is benign by adding additional edges to each node. In the second stage, which is the main part of the algorithm, we continuously increase the graph's conductance to random walks while assuring that it stays benign. Finally, in the last stage, we exploit the graph's logarithmic diameter to construct a well-formed tree in $O(\log n)$ time. This phase primarily uses techniques from [AS03, AW07, GHSS17, GHS19b]. We now describe each phase in more detail.

(Step 1) Initialization: Before the first iteration, we need to prepare the initial communication graph to comply with parameters Δ and Λ, i.e., we must turn it into a benign graph. W.l.o.g., we can assume that G is a simple bidirected graph. Otherwise, all of its multi-edges can be merged into a single edge, and all directed edges $(v, w) \in E$ can be bi-directed by sending v's identifier to w. Next, we deal with the graph's regularity. Arguably, the easiest way to turn a graph of maximum degree Δ_G into regular graph is adding $\Delta_G - \mathsf{deg}(v)$ self-loops to each node $v \in V$, so all nodes have the same degree. However, this is not possible, as in our model, each node can only send and receive $O(\mathsf{deg}(v) \log n)$ messages per round, which might be lower than Δ_G. Thus, we need another approach. Instead, we use the concept of *virtual nodes* V', which are simulated by the nodes of graph G. A virtual node $v' \in V'$ is simulated by $v \in V$ and has its own virtual identifier of size $O(\log n)$. This virtual identifier consists of the original node's identifier combined with a locally unique identifier. For the simulation, any message intended for v' will first be sent to v (using v's identifier) and then locally passed to v'. Given this concept, we show that the node in graph $G := (V, E)$ can simulate a graph $G' := (V', E')$ with $|V'| = 2|E|$ that only consists of virtual nodes and edges between them. We construct V' through the following process:

1. First, For each edge $\{v, w\} \in E$ both v and w create and simulate virtual nodes v' and w' and add them to V'.

2. Second, v and w connect the virtual nodes v' and w' via an edge $\{v', w'\}$ by exchanging the respective identifiers. This ensures that for each edges $(v, w) \in E$, there is an edge $(v', w') \in E'$.

3. Finally, all virtual nodes of a real node v are connected in a cycle. For this, v first sorts its virtual nodes in an arbitrary order $v'_1, \dots, v'_{\mathsf{deg}(v)}$. Then, it adds bidirected edges between v_1 and $v_{\mathsf{deg}(v)}$ and every consecutive pair of nodes in the order.

The resulting graph G' is connected and each node $v' \in V'$ has at most 3 edges. To be precise, it has exactly one edge to another virtual node $w' \in V'$ simulated by $v \neq w \in V$ and at most two connections to the predecessor and successor in the cycle of virtual nodes simulated by v. Further, each original node simulates exactly $\mathsf{deg}(v)$ virtual nodes, so as long as each virtual node receives at most $O(\log n)$ messages (which is the case in our algorithm), this is possible in our model. Later, we will show how to revert the simulation and obtain a well-formed tree for the original node set V.

Given that the graph is regular, we need to increase its degree and minimal cut. Since the input graph has a maximum degree of 3, this is quite simple as we can assume $6\Lambda \leq \Delta$ by choosing Δ big enough. Given this assumption, the graph can be turned benign in 2 steps:

1. First, all edges are copied Λ times to obtain the desired minimum cut. After this step, each node has at least Λ edges to other nodes and at least Λ edges cross each cut.

2. Then, each node adds self-loops until its degree is Δ and each node has $\frac{\Delta}{2}$ self-loops. As we chose $6\Lambda \leq \Delta$, this is always possible.

Thus, the resulting graph is benign. Further, note that the resulting graph is a multi-graph while G is a simple graph.

(Step 2) Construction of a Low Diameter Overlay: Let now $G_0 = (V', E_0)$ be the resulting benign graph. The algorithm proceeds in iterations $1, \dots, L$. In each iteration, a new communication graph $G_i = (V', E_i)$ is created through sampling $\frac{\Delta}{8}$ new neighbors via random walks of length ℓ. Each node $v \in V'$ creates $\frac{\Delta}{8}$ messages containing its own identifier, which we call *tokens*. Each token is randomly forwarded for ℓ rounds in G_i. More precisely, each node that receives a token picks one of its incident edges in G_i uniformly at random and sends the token to the corresponding node.[1] This happens independently for each token. If v receives less than $\frac{3}{8}\Delta$ tokens after ℓ steps, it sends its identifier back to all the tokens' origins to create a bidirected edge. Otherwise, it picks $\frac{3}{8}\Delta$ tokens at random (without replacement)[2]. Since the origin's identifier is stored in the token, both cases can be handled in one communication round. Finally, each node adds self-loops until its degree is Δ again. The whole procedure is given in Figure 2.1 as the method CreateExpander$(G_0, \ell, \Delta, \Lambda, L)$. The subroutine MakeBenign$(G_0, \ell, \Delta, \Lambda)$ add edges and self-loops to make the graph comply with Definition 2.1 (i.e., it implements the first stage). Our main observation is that

```
CreateExpander($(G_0, \ell, \Delta, \Lambda, L)$):
Each node $v \in V'$ executes:

  1. $E_0 \longleftarrow$ MakeBenign($(G_0, \ell, \Delta, \Lambda)$)
  2. For $i = 0, \dots, L$:
     (a) Create $\Delta/8$ tokens that contain $v$'s identifier and store them in $T_0$.
     (b) For $j = 1, \dots, \ell$:
           Independently send each token from $T_{j-1}$ along a random incident edge in $G_i = (V', E_i)$.
           Store all received tokens in the buffer $T_j$.
     (c) Pick (up to) $3\Delta/8$ tokens $w_1, \dots, w_{\Delta'}$ from $T_\ell$ without replacement.
     (d) Create edges $E_{i+1} := \{\{v, w_1\}, \dots, \{v, w_{\Delta'}\}\}$ by sending $v$'s identifier to each $w_j$.
     (e) Add self-loops $\{v, v\}$ to $E_{i+1}$ until $|E_{i+1}| = \Delta$.
```

Figure 2.1: Pseudocode for the CreateExpander algorithm that turns any initial overlay into an expander.

after $L = O(\log n)$ iterations, the resulting graph G_L has constant conductance, w.h.p., which implies that its diameter is $O(\log n)$. Furthermore, the degree of G_L is $O(\Delta)$ by construction. Finally, if we add any virtual nodes in the first stage, we can now merge them back into a single node (with all connections of all its virtual nodes). For this, we simply transform each edge $(v', w') \in E_L$ between two virtual nodes v', w' to an edge (v, w) between two original nodes $v, w \in V$. This produces a graph with the same degree distribution as G and can only decrease the diameter further. In particular, the degree of node $v \in V$ in the resulting graph is $O(\mathsf{deg}_G(v) \cdot \Delta)$ which is within $O(\mathsf{deg}_G(v) \cdot \log n)$ as $\Delta \in O(\log n)$. Following this argument, the resulting graph has degree $O(\Delta_G \cdot \log n)$. We denote this graph as G'_L.

[1] We will show that each node only sends and receives at most $O(\Delta)$ tokens in each round, w.h.p.
[2] We will see, however, that this case does not occur w.h.p.

(Step 3) Finalization: To obtain a well-formed tree T_G, we perform a BFS on G'_L starting from the node with the lowest identifier. Since a node cannot locally check whether it has the lowest identifier, implementing this step is slightly more complex. The algorithm proceeds for $L \in O(\log n)$ rounds. In the first round, every node creates a token message that contains its identifier. Then, it sends the token to all its neighbors. For all remaining rounds $1, \dots, L$, every node that receives one or more tokens only forwards the token with the lowest identifier to all its neighbors and drops all others. Since the graph's diameter is $O(\log n)$, all nodes must have received the lowest identifier at least once after these $L \in O(\log n)$ rounds. Finally, each node v marks the edge (v, w) over which it first received the token with the lowest identifier. Ties can be broken arbitrarily. If the node itself has the smallest identifier, it does not mark any edge. All marked edges then constitute a tree T with degree $O(\Delta_G \cdot \log n)$ and diameter $O(\log n)$. Note that this process requires $O(\mathsf{deg}_G(v) \cdot \log n)$ messages per node and round, as each node sends at most one token to all its neighbors per round. To transform this tree T into a well-formed tree, we perform the *merging step* of the algorithm of [GHSS17, Theorem 2]. This deterministic subroutine transforms any tree into a well-formed tree of degree $O(\log n)$ in $O(\log n)$ rounds.

To make this thesis self-contained, we sketch the approach of [GHSS17, Theorem 2] in the remainder: The algorithm first transforms T into a constant-degree *child-sibling tree* [AW07], in which each node arranges its children as a path and only keeps an edge to one of them. For each inner node $w \in V$ let $w_1, \dots, w_{\mathsf{deg}_{G'_L}(w)-1}$ denote its children in T sorted by increasing identifier. Now w only keeps the child with the lowest identifier and delegates the others as follows: Each $w_i \in N(w)$ with $i > 1$ changes its parent to be its predecessor w_{i-1} on the path and stores its successor w_{i+1} as a sibling (if it exists). In the resulting tree, each node stores at most three identifiers: a parent and possibly a sibling and a child. Since we can interpret the sibling of a node as a second child, we obtain a binary tree. This transformation takes $O(1)$ rounds and requires $O(\mathsf{deg}_G(v) \cdot \log n)$ communication as each node v needs to send two identifiers to its children.

Note that the tree's diameter has now been increased by a factor of $O(\Delta_G \cdot \log n)$. Based on this binary tree, we construct a ring of virtual nodes using the so-called *Euler tour technique* (see, e.g., [TV85, GHSS17, FHS20]). Consider the depth-first traversal of the tree that visits the children of each node in order of increasing identifier. A node occurs at most three times in this traversal. Let each node act as a distinct virtual node for each such occurrence and let $k \leq 3n$ be the number of virtual nodes. More specifically, every node v executes the following steps:

1. v creates virtual nodes $v^0, \dots, v^{\mathsf{deg}(v)-1}$ where v^i has the virtual identifier $id(v^i) := v \circ i$. Intuitively, the node v^0 corresponds to the traversal visiting v from its parent. Analogously, each v^i is the visit from child w_i.

2. v sends the identifier of v^0 and $v^{\mathsf{deg}(v)-1}$ to its parent. Note that v^0 and $v^{\mathsf{deg}(w)-1}$ may be the same virtual node if v has no children.

3. Let w_i^0 and $w_i^{\mathsf{deg}(w)-1}$ be the identifier received from w_i, i.e., the i^{th} child of v. Then v sets w_i^0 as the successor of v^{i-1} and $w_i^{\mathsf{deg}(w)-1}$ as the predecessor of v^i. In other words, v^{i-1} and v^i are connected to the first and last virtual node of w_i.

4. Finally, each virtual node introduces itself to its predecessor and successor.

Therefore, the nodes can connect their virtual nodes into a ring in $O(1)$ rounds by sending at most two messages per edge in each round. Next, we use the *Pointer jumping technique* (see, e.g., [TV85, GHSS17, FHS20]) to quickly add *chords* (i.e., shortcut edges) to the ring. To be precise, the virtual nodes execute the following protocol for $L \in O(\log n)$ rounds:

1. Let l_0 and r_0 be the predecessor and successor of v in the ring. In the first round of pointer jumping, v sends l_0 to r_0 and vice versa.

2. In round $t > 0$, each node receives an identifier l_{t-1} and r_{t-1} sent in the previous round. It sets l_t to the identifier received from l_{t-1} and r_t to the identifier received from r_{t-1}. Finally, it sends l_t to r_t and vice versa.

A simple induction reveals that the distance between these neighbors (w.r.t the ring) doubles from round to round (until the distance exceeds the number of virtual nodes k). Based on this observation, it is not hard to show that after the L rounds, the graph's diameter has reduced to $O(\log n)$ while the degree has grown to $O(\log n)$. A final BFS from the node of the lowest identifier then yields our desired well-formed tree T_G, which concludes the algorithm.

With further techniques that exploit the structure of the chords, the degree can be reduced to 6. For details, we refer to [GHSS17, Theorem 2]. Another possibility to achieve a constant degree is the algorithm by Aspnes and Wu [AW07]. This algorithm requires a graph of outdegree 1 — which simply is a by-product of the BFS — and requires $O(W + \log n)$ time, w.h.p. The term W denotes the length of the node identifiers, which is also $O(\log n)$ in our case. Although the algorithm is simple, elegant, and has the desired runtime, we do not use it as a black box as it would lead to problems with some applications. We need to create a well-formed tree for subgraphs with $n' < n$ nodes for some of the applications and require that the runtime is logarithmic in n' and not n. Without additional analysis, the algorithm by Aspnes and Wu would still take $O(\log n)$ time, w.h.p., whereas the approach sketched above always finishes in $O(\log n')$ time.

2.2 Mathematical Preliminaries for Theorem 1

In this section, we establish some mathematical preliminaries that we require for the analysis of our algorithm. First, we present some results on random walks on regular graphs. Second, we present a variant of the union bound that helps us when considering a superpolynomial number of events.

2.2.1 Random Walks on Regular Graphs

In this section, we observe the behavior of (short) random walks on regular graphs and establish useful definitions and results. Starting with the most basic definition, a random walk on a graph $G := (V, E)$ is a stochastic process $(v_i)_{i \in \mathbb{N}}$ that starts at some node $v_0 \in V$ and in each step moves to some neighbor of the current node. If G is Δ-regular, the probability of moving from v to its neighbor w is $e(v,w)/\Delta$. Here, $e(u, w)$ denotes the number of edges between u and w as G is a multigraph. We say that a Δ-regular graph G (and the corresponding walk on its nodes) is *lazy* if each node has at least $\frac{\Delta}{2}$ self-loops (and therefore a random walk stays at its current node with probability at least $1/2$ in each step). Further, we define $G^\ell := (V, E_\ell)$ to be the ℓ-walk graph of G, i.e., G^ℓ is the multigraph where each edge $(v,w) \in E_\ell$ corresponds to an ℓ-step *walk* in G. Note that a walk can visit the same edge more than once, so G^ℓ must be a multigraph. For $v, w \in V$ let $X_v^\ell(w,G)$ be the indicator for the event that an ℓ-step random walk in G which started in v ends in w. Analogously, let $X_v^1(w,G^\ell)$ be the indicator that a 1-step random walk in G^ℓ which started in v ends in w. If we consider a fixed node v that is clear from the context, we may drop the subscript and write $X^1(w,G^\ell)$ instead. Further, let $P_\ell(v,w)$ be the exact number of walks of length ℓ between v and w in G. Note that it holds $P_1(v,w) = e(v,w)$. Given these definitions, the probability to move from v to w in G^ℓ is given by the following lemma:

Lemma 2.1. *Let G be a Δ-regular graph and G^ℓ its ℓ-walk graph for some $\ell > 1$, then it holds:*

$$\mathbf{Pr}\big[X^1(w,G^\ell)\big] = \mathbf{Pr}\big[X^\ell(w,G)\big] = \frac{P_\ell(v,w)}{\Delta^\ell} \tag{2.1}$$

Proof. The statement can be proved via an induction over ℓ, the length of the walk. For the base case, we need to show that 1-step random walk in G is equivalent to picking an outgoing edge in $G^1 := G$ uniformly at random. This follows trivially from the definition of a random walk. Now suppose that performing an $(\ell-1)$-step random walk in G is equivalent to performing a 1-step walk in $G^{\ell-1}$. Consider a node $w \in V$ and let N_w denote its neighbors in G *and* $w$itself. By the law of total probability, it holds:

$$\mathbf{Pr}\big[X^\ell(w,G)=1\big] := \sum_{u\in N_w} \mathbf{Pr}\big[X^{\ell-1}(u,G)=1\big] \cdot \mathbf{Pr}\big[X^\ell(w,G)=1 \mid X^{\ell-1}(u,G)=1\big] \tag{2.2}$$

Using the *induction hypothesis* we we can substitute $\mathbf{Pr}\big[X^{\ell-1}(u,G)=1\big]$, the probability of a $\ell-1$-step walk in G, for $\mathbf{Pr}\big[X^1(u,G^{\ell-1})=1\big]$, the probability of single step in $G^{\ell-1}$, and get:

$$\mathbf{Pr}\big[X^\ell(w,G)=1\big] := \sum_{u\in N_w} \mathbf{Pr}\big[X^{\ell-1}(u,G)=1\big] \cdot \mathbf{Pr}\big[X^\ell(w,G)=1 \mid X^{\ell-1}(u,G)=1\big] \tag{2.3}$$

$$= \sum_{u\in N_w} \frac{P_{\ell-1}(v,u)}{\Delta^{\ell-1}} \cdot \mathbf{Pr}\big[X^\ell(w,G)=1 \mid X^{\ell-1}(u,G)=1\big] \tag{2.4}$$

Recall that G is a multigraph, and there can be more than one edge between each u and w and $e(u,w)$ denote the number of edges between u and w for every $u \in N_w$. Since we defined that $w \in N_w$, the value $e(w,w)$ counts w's self-loops. Since G is Δ-regular, the probability that a random walk at node u moves to w is exactly $\frac{e(u,w)}{\Delta}$. Back in the formula, we get:

$$\mathbf{Pr}\big[X^\ell(w,G)=1\big] = \sum_{u\in N_w} \frac{P_{\ell-1}(v,u)}{\Delta^{\ell-1}} \cdot \mathbf{Pr}\big[X^\ell(w,G)=1 \mid X^{\ell-1}(u,G)=1\big] \tag{2.5}$$

$$= \sum_{u\in N_w} \frac{P_{\ell-1}(v,u)}{\Delta^{\ell-1}} \frac{e(u,w)}{\Delta} = \frac{1}{\Delta^\ell} \sum_{u\in N_w} P_{\ell-1}(v,u) \cdot e(u,w) \tag{2.6}$$

Finally, note that $\sum_{u \in N_w} P_{\ell-1}(v,u) \cdot e(u,w)$ counts all paths of length exactly ℓ from v to w in G. This follows because each path $P := (e_1, \ldots, e_\ell)$ from u to w can be decomposed into a path $P' := (e_1, \ldots, e_{\ell-1})$ of length $\ell - 1$ to some neighbor of w (or w itself) and the final edge (or self-loop) e_ℓ. Thus, it follows that:

$$\mathbf{Pr}\big[X^\ell(w,G) = 1\big] = \frac{1}{\Delta^\ell} \sum_{u \in N_w} P_{\ell-1}(v,u) \cdot e(u,w) = \frac{P_\ell(v,w)}{\Delta^\ell} = \mathbf{Pr}\big[X^1(w,G^\ell) = 1\big] \tag{2.7}$$

This was to be shown.

□

In other words, the multigraph G^ℓ is Δ^ℓ-regular and has edge (v,w) for every walk of length ℓ between v and w.

We further need two well-known facts about the conductance. First, we see that we can relate the minimum conductance of a Δ-regular graph to its minimum cut. It holds:

Lemma 2.2 (Minimum Conductance). *Let $G := (V,E)$ be any Δ-regular connected graph with minimum cut $\Lambda \geq 1$. Then for all $\delta \in (0,1]$ it holds:*

$$\Phi_\delta(G) \geq \frac{2\Lambda}{\Delta \delta n}. \tag{2.8}$$

Proof. Consider the set S with $|S| \leq \frac{\delta n}{2}$ that minimizes $\Phi(S')$ among all sets $|S'| \leq \frac{\delta}{2} n$. Then, it holds by the definition $\Phi_\delta(G)$, S and Λ that:

$$\Phi_\delta(G) := \min_{S' \subset V, |S'| \leq \frac{\delta |V|}{2}} \Phi(S') := \Phi(S) := \frac{O_S}{\Delta |S|} \tag{2.9}$$

$$\geq \frac{2 O_S}{\Delta \delta n} \qquad \triangleright \textit{As } |S| \leq \frac{\delta}{2} n \tag{2.10}$$

$$\geq \frac{2\Lambda}{\Delta \delta n} \qquad \triangleright \textit{As } |O_S| \geq \Lambda \tag{2.11}$$

□

Second, we show that a constant conductance implies a logarithmic diameter if the graph is regular. It holds:

Lemma 2.3 (High Conductance implies Low Diameter). *Let $G := (V,E)$ be any lazy bi-directed Δ-regular graph with conductance Φ, then the diameter of G is at most $O(\Phi^{-2} \log n)$.*

Proof. We will prove this lemma by analyzing the distribution of random walks on G. Let $v, w \in V$ be two nodes of G and let $\ell > 0$ be an integer. We denote $p^\ell(v,w) \in [0,1]$ as the probability that an ℓ-step random walk that starts in v, ends in w. Note that $p^\ell(v,w) > 0$ implies that there must exist path of length ℓ from v to w. Following this argument, if it holds $p^\ell(v,w) > 0$ for all pairs $v, w \in V$, then the graph's diameter must be smaller or equal to ℓ. Thus, in the following, we will show that for $\ell \in \Omega(\Phi_G^{-2} \log n)$ we have $p^\ell(v,w) > 0$ for all pairs of nodes. First, we note that a sharp upper bound on the probabilities also implies a lower bound.

Claim 1. *Let $v \in V$ be a node with $p^\ell(v,w) \leq \frac{1}{n} + \frac{1}{n^2}$ for all $w \in V$. Then, it holds*

$$p^\ell(v,w) \geq \frac{1}{n^2} \tag{2.12}$$

Proof. As each random walk must end some node $w \in V$, we have:

$$\sum_{w \in V} p^\ell(v,w) = 1 \tag{2.13}$$

Together with our upper bound of $\frac{1}{n} + \frac{1}{n^2}$, we can now derive the following lower bound

$$p^\ell(v,w) = 1 - \sum_{u \in V \setminus \{w\}} p^\ell(v,u) \geq 1 - \sum_{u \in V \setminus \{w\}} \left(\frac{1}{n} + \frac{1}{n^2}\right) \tag{2.14}$$

$$= 1 - \left(\frac{n-1}{n} + \frac{n-1}{n^2}\right) = 1 - \left(1 - \frac{1}{n} + \frac{1}{n} - \frac{1}{n^2}\right) = \frac{1}{n^2} \tag{2.15}$$

□

Thus, a low enough maximal probability implies a positive lower bound. Our next goal is to find such a precise upper bound. We will use well-known concepts from the analysis of Markov chains to do this. We define $\pi := (\pi_v)_{v \in V}$ as the stationary distribution of a random walk on G. For any Δ-regular graph, it holds:

$$\pi_v := \frac{d_v}{|E|} = \frac{\Delta}{\Delta n} = \frac{1}{n} \tag{2.16}$$

For a connected, bidirected, non-bipartite graph, the distribution of possible endpoints of a random walk converges towards its stationary distribution. For a fixed ℓ, we define the *relative pointwise distance* as:

$$\rho_G(\ell) := \max_{v,w \in V} \left\{ \frac{p^\ell(v,w) - \pi_w}{\pi_w} \right\} \tag{2.17}$$

This definition describes how *far* the distribution is from the stationary distribution after ℓ steps. Given this definition, it is easy to see that the following claim holds:

Claim 2. *Suppose that $\rho_G(\ell) < \frac{1}{n}$, then it holds for all $v, w \in V$ that*

$$p^\ell(v,w) \leq \frac{1}{n} + \frac{1}{n^2} \tag{2.18}$$

Proof. For contradiction, assume that the statement is false. Then, there is a pair $v', w' \in V$ with

$$p^\ell(v',w') = \frac{1}{n} + \frac{c}{n^2} \tag{2.19}$$

for some $c > 1$. Further, it must hold that

$$\rho_G(\ell) := \max_{v,w \in V} \left\{ \frac{p^{\ell}(v,w) - \pi_w}{\pi_w} \right\} \geq \frac{p^{\ell}(v',w') - \pi_{v,w}}{\pi_{v,w}} \tag{2.20}$$

$$= \frac{p^{\ell}(v',w') - \frac{1}{n}}{\frac{1}{n}} = \frac{\frac{1}{n} + \frac{c}{n^2} - \frac{1}{n}}{\frac{1}{n}} = \frac{c}{n} > \frac{1}{n} > \rho_G(\ell). \tag{2.21}$$

This is the desired contradiction. □

Thus, we will determine an upper bound for $\rho_G(\ell)$ in the remainder. In an influential article, Jerrum and Sinclair proved that *relative pointwise distance* after ℓ steps is closely tied to the graph's conductance. In particular, they showed that:

Lemma 2.4 (Theorem 3.4 in [SJ89], simplified). *Let G be lazy, regular, and connected. Further, let π be its stationary distribution of a random walk. Then for any node $v \in V$, the relative pointwise distance satisfies*

$$\rho_G(\ell) \leq \frac{\left(1 - \frac{\Phi_G^2}{2}\right)^{\ell}}{\pi^*} \tag{2.22}$$

With $\pi^* := \max_{v \in V}\{\pi_v\}$

In the original lemma, the underlying Markov chain must be ergodic, i.e., every state is reachable from every other state, and time-reversible, i.e., it holds $p^{\ell}(v,w) = p^{\ell}(w,v)$. The first property is implied by the fact that the graph is connected, so every node is reachable. The latter follows from the facts that the graph is bi-directed and regular, so it holds $p_1(v,w) = \frac{e(v,w)}{\Delta} = \frac{e(w,v)}{\Delta} = p_1(w,v)$. Further, note that for our graph, we have $\pi^* := \frac{1}{n}$. Plugging this and $\ell := 4\Phi^{-2}\log n$ in the formula, we get:

$$\rho_G(\ell) \leq \frac{\left(1 - \frac{\Phi_G^2}{2}\right)^{\ell}}{\pi_v} = n\left(1 - \frac{\Phi_G^2}{2}\right)^{\ell} = n\left(1 - \frac{\Phi_G^2}{2}\right)^{4\Phi_G^{-2}\log n} \tag{2.23}$$

$$\leq ne^{-2\log n} \leq \frac{1}{n}. \tag{2.24}$$

Here, Inequality (2.24) follows from the well-known fact that $(1 - 1/x)^x \leq 1/e$ for any $x > 1.5$, which clearly holds as $\Phi_G^2 < 1$. Thus, following Claims 1 and 2, all probabilities are strictly positive after ℓ steps of the random walk and the diameter must be smaller ℓ. □

For our analysis, it will be crucial to observe the (small-set) conductance of G^{ℓ} for a constant ℓ. However, the standard Cheeger inequality (see, e.g., [Sin12] for an overview) that is most commonly used to bound a graph's conductance with the help of the graph's eigenvalues does not help us in deriving a meaningful lower bound for $\Phi_{G^{\ell}}$. In particular, it only states that $\Phi_{G^{\ell}} = \Theta(\ell\Phi_G^2)$. Thus, it only provides a useful bound if $\ell = \Omega(\Phi_G^{-1})$, which is too big for our purposes, as $\Omega(\Phi_G^{-1})$ is only constant if Φ_G is constant. More recent Cheeger inequalities shown in [LGT14] relate the conductance of smaller subsets to higher eigenvalues of the random walk matrix. At first glance, this seems helpful, as one could use these to show that at least the small sets start to be more densely connected and then, inductively, continue the argument. Still, even with this approach, constant length walks are out of the question as these new Cheeger inequalities introduce an additional tight $O(\log n)$ factor in the approximation for these small sets. Thus, the random walks would need to be of length $\Omega(\log n)$, which is still too much to achieve our bounds. Instead, we use the following result by Kwok and Lau [KL14], which states that Φ_{G^ℓ} improves even for constant values of ℓ. Given this bound, we can show that benign graphs increase their (expected) conductance from iteration to iteration. In the following, we prove Lemma 2.5 by outlining the proofs of Theorem 1 and 3 in [KL14]. It holds that:

Lemma 2.5 (Conductance of G^ℓ, Based on Theorem 1 and 3 in [KL14]). *Let $G = (V, E)$ be any connected Δ-regular lazy graph with conductance Φ_G and let G^ℓ be its ℓ-walk graph. For a set $S \subset G$ define $\Phi_{G^\ell}(S)$ as the conductance of S in G^ℓ. Then, it holds:*

$$\Phi_{G^\ell}(S) \geq \min\left\{\frac{1}{2}, \frac{1}{40}\sqrt{\ell}\Phi(G)\right\} \tag{2.25}$$

Further, if $|S| \leq \delta n$ for any $\delta \in (0, \frac{1}{2}]$, we have

$$\Phi_{G^\ell}(S) \geq \min\left\{\frac{1}{4}, \frac{1}{40}\sqrt{\ell}\Phi_{2\delta}(G)\right\} \tag{2.26}$$

Proof. Before we go into the details, we need another batch of definitions from the study of random walks and Markov chains. Let $G := (V, E)$ be a Δ-regular, lazy graph and let $A_G \in \mathbb{R}^{n \times n}$ the stochastic random walk matrix of G. Each entry $A_G(v, w)$ in the matrix has the value $\frac{e(v,w)}{\Delta}$ where $e(v, w)$ denotes the number of edges between v and w (or self-loops if $v = w$). Likewise A_G^ℓ is the random walk matrix of G^ℓ where each entry has value $\frac{P_\ell(v,w)}{\Delta^\ell}$. Note that both A_G and A_G^ℓ are doubly-stochastic, meaning that their rows and columns sum up to 1. For these types of weighted matrices, Kwok and Lau define the *expansion* $\varphi(S)$ of a subset $S \subset V$ as follows:

$$\varphi(S) = \frac{1}{|S|} \sum_{v \in S, w \in \overline{S}} A_G(v, w) \tag{2.27}$$

For regular graphs (and *only* those), this value is equal to the conductance $\Phi_G(S)$ of S, which we observed before. The following elementary calculation can verify this claim:

$$\varphi(S) = \frac{1}{|S|} \sum_{v \in S, w \in \overline{S}} A_G(v, w) = \frac{1}{|S|} \sum_{v \in S, w \in \overline{S}} \frac{e(v, w)}{\Delta} \tag{2.28}$$

$$= \frac{\sum_{v \in S, w \in \overline{S}} e(v, w)}{\Delta |S|} =: \Phi_G(S) \tag{2.29}$$

Therefore, the claim that Kwok and Lau make for the expansion also holds for the conductance of regular graphs[3]. The proof in [KL14] is based on the function $C^{(\ell)}(|S|)$ introduced by Lovász and Simonovits[LS90].

[3]Indeed, they explicitly mention that for non-regular graph one could define a *escape probability* for which their claims would hold and which could be used instead of the conductance in our proofs. Nevertheless, since we only observe regular graphs, we use the notion of conductance to avoid introducing more concepts.

Consider a set $S \subset V$, then Lovasz and Simonovits define the following curve that bounds the distribution of random walk probabilities for the nodes of S.

$$C^{(\ell)}(|S|) = \max_{\delta_0+\cdots+\delta_n = x, 0 \leq \delta_i \leq 1} \sum_{i=1}^{n} \delta_i (A^{\ell} p_S)_i \tag{2.30}$$

Here, the vector p_S is the so-called characteristic vector of S with $p_i = \frac{1}{|S|}$ for each $v_i \in S$ and 0 otherwise. Further, the term $(A^{\ell}p)_i$ denotes the i^{th} value of the vector $A^{\ell}p_S$. Lovász and Simonovits used this curve to analyze the mixing time of Markov chains. Kwok and Lau now noticed that it also holds that:

Lemma 2.6 (Lemma 6 in [KL14]). *It holds:*

$$\Phi_{G^{\ell}}(S) \geq 1 - C^{(\ell)}(|S|) \tag{2.31}$$

Based on this observation, they deduce that a bound for $1 - C^{(\ell)}(S)$ doubles as a bound for $\Phi_{G^{\ell}}$. In particular, they can show the following bounds for $C^{(\ell)}(|S|)$:

Lemma 2.7 (Lemma 7 in [KL14]). *It holds*

$$C^{(\ell)}(|S|) \leq 1 - \frac{1}{20}\left(1 - (1 - \Phi_G)^{\sqrt{\ell}}\right) \tag{2.32}$$

We refer the interested reader to Lemma 7 of [KL14] for the full proof with all necessary details. For the next step, we need the following well-known inequality:

Lemma 2.8. *For any $t > 1$ and $z \leq \frac{1}{2}$, it holds:*

$$(1 - z)^t \leq 1 - \frac{1}{2}zt \tag{2.33}$$

Now assume that G does not already have a *constant* conductance of $\Phi(G) = \frac{1}{2}$. Plugging this assumption and the two insights by Kwok and Lau together, we get

$$\begin{aligned}
\Phi_{G^{\ell}}(S) &\geq 1 - C^{(\ell)}(|S|) && \triangleright \textit{By Lemma 2.6} \quad (2.34)\\
&\geq \frac{1}{20}\left(1 - (1 - \Phi_G)^{\sqrt{\ell}}\right) && \triangleright \textit{By Lemma 2.7} \quad (2.35)\\
&\geq \frac{1}{20}\left(1 - (1 - \frac{1}{2}\sqrt{\ell}\Phi_G)\right) = \frac{\sqrt{\ell}}{40}\Phi_G && \quad (2.36)
\end{aligned}$$

The last inequality follows from the fact that $\sqrt{\ell}\Phi_G$ is at most $\frac{1}{2}$. The second part of the theorem can be derived similarly. Again, we observe an auxiliary lemma by Kwok and Lau and see:

Lemma 2.9 (Lemma 10 in [KL14]). *Let S be set of size at most δn with $\delta \in [0, \frac{1}{4})$. Then, it holds:*

$$C^{(\ell)}(|S|) \leq 1 - \frac{1}{20}\left(1 - (1 - 2\Phi_{2\delta}(G))^{\sqrt{\ell}}\right) \tag{2.37}$$

Proof. Analogously to the previous case, we get for $\Phi_{2\delta}(G) \leq \frac{1}{4}$ that

$$\begin{aligned}
\Phi_{G^\ell}(S) &\geq 1 - C^{(\ell)}(|S|) && \triangleright \textit{By Lemma 2.6} && (2.38)\\
&\geq \frac{1}{20}\left(1 - (1 - 2\Phi_{2\delta}(G))^{\sqrt{\ell}}\right) && \triangleright \textit{By Lemma 2.9} && (2.39)\\
&\geq \frac{1}{20}\left(1 - (1 - 2\Phi_{2\delta}(G))\right) && && (2.40)\\
&= \frac{\sqrt{\ell}}{20}\Phi_{2\delta}(G) \leq \frac{\sqrt{\ell}}{40}\Phi_{2\delta}(G) && && (2.41)
\end{aligned}$$

In the last inequality, we used Lemma 2.8 with $z = 2\Phi_{2\delta}(G)$. □

Finally, these two lower bounds are too loose for graphs (and subsets) that already have good conductance. Instead we require that $\Phi_{G^\ell}(S)$ is *at least as* big as $\Phi_G(S)$. Note that this is not necessarily the case for all graphs. Instead, we must use the fact that our graphs are *lazy*. We show this in the following lemma:

Lemma 2.10. *Let $G := (V, E)$ be any connected Δ-regular lazy graph with conductance Φ_G and let G^ℓ be its ℓ-walk graph. For a set $S \subset G$ define $\Phi_{G^\ell}(S)$ the conductance of S in G^ℓ. Then, it holds:*

$$\Phi_{G^\ell}(S) \geq \Phi_G(S) \tag{2.42}$$

Proof. Our technical argument is based on the following recursive relation between $C^{(\ell+1)}$ and $C^{(\ell)}$, which was (in part) already shown in [LS90]:

Lemma 2.11 (Lemma 1.4 in [LS90]). *It holds*

$$C^{(\ell+1)}(|S|) \leq \frac{1}{2}\left(C^{(\ell)}(|S| + 2\Phi_G|\hat{S}|) + C^{(\ell)}(|S| - 2\Phi_G|\hat{S}|)\right) \tag{2.43}$$

Here, we use the abbreviation $|\hat{S}| := \max\{|S|, n - |S|\}$. The remainder of the proof is based on two claims. First, we claim that $C^{(\ell)}(|S|)$ is monotonically increasing in ℓ.

Claim 3. *It holds $C^{(\ell)}(|S|) \leq C^{(\ell-1)}(|S|)$*

Proof. This fact was already remarked in [LS90] based on an alternative formulation. However, given that $C^{(\ell)}$ is concave, it holds that for all values $\gamma, \beta \geq 0$ with $\gamma \leq \beta$ that

$$C^{(\ell)}(S + \beta\hat{S}) + C^{(\ell)}(|S| - \beta\hat{S}) \leq C^{(\ell)}(S + \gamma|\hat{S}|) + C^{(\ell)}(|S| - \gamma|\hat{S}|) \tag{2.44}$$

And thus, together with Lemma 2.11, we get:

$$\begin{aligned}
C^{(\ell)}(|S|) &\leq \frac{1}{2}\left(C^{(\ell-1)}(|S| + 2\Phi_G|\hat{S}|) + C^{(\ell-1)}(|S| - 2\Phi_G|\hat{S}|)\right) && (2.45)\\
&\leq \frac{1}{2}\left(C^{(\ell-1)}(|S| + 0 \cdot |\hat{S}|) + C^{(\ell-1)}(|S| - 0 \cdot |\hat{S}|)\right) && (2.46)\\
&= C^{(\ell-1)}(|S|) && (2.47)
\end{aligned}$$

Here, we chose $\beta = 2\Phi_G$ and $\gamma = 0$ and applied Equation 2.44. This proves the first claim. $\square$

Second, we claim that $C^{(1))}(|S|)$ is *equal* to $1 - \Phi_G(S)$ as long as the graph we observe is lazy.

Claim 4. *It holds* $C^{(1)}(|S|) = 1 - \Phi_G(S)$

Proof. For this claim (which was not explicitly shown in [KL14], but implied in [LS90]) we observe

$$C^{(1)}(S) = \max_{\delta_0 + \dots + \delta_n = x, 0 \leq \delta_i \leq 1} \sum_{i=1}^{n} \delta_i (A_G p_S)_i \tag{2.48}$$

and find the assignment of the δ's that maximizes the sum. Lovasz and Simonovits already remarked that it is maximized by setting $\delta_i = 1$ for all $v_i \in S$. However, we prove it here since there is no explicit lemma or proof to point to in [LS90]. First, we show that all entries $(A_G p_S)_i$ for nodes $v_i \in S$ are least $\frac{1}{2|S|}$ and all entries $(A_G p_S)_{i'}$ for nodes $v_{i'} \notin S$ are at most $\frac{1}{2|S|}$. We begin with the nodes in S. Given that G is Δ-regular and lazy, we have for all $v_i \in S$ that

$$(A_G p_S)_i = \sum_{j=1}^{n} A_G(v_i, v_j) p_{S_j} \tag{2.49}$$

$$\geq A_G(v_i, v_i) p_{S_i} \tag{2.50}$$

$$\geq \frac{1}{2|S|}. \tag{2.51}$$

Here, $p_{S_i} = \frac{1}{|S|}$ follows because $v_i \in S$ per definition. The inequality $A_G(v_i, v_i) \geq \frac{1}{2}$ follows from the fact that A is lazy and each node has a self-loop with probability $\frac{1}{2}$. As a result, the entry $(A_G p_S)_i$ for $v_i \in S$ has at least a value of $\frac{1}{2|S|}$, even if it has no neighbors in S. On the other hand, we have for all nodes $v_{i'} \notin S$ that

$$(A_G p_S)_{i'} = \sum_{j=1}^{n} A_G(v_j, v_{i'}) p_j \tag{2.52}$$

$$= \sum_{v_j \in S} A_G(v_{i'}, v_j) \frac{1}{|S|} \tag{2.53}$$

This follows from excluding all entries p_j with $v_j \notin S$. Note that for these values, it holds $p_j = 0$. Further, Since A is Δ-regular and lazy, each node $v_{i'} \notin S$ has at most $\frac{\Delta}{2}$ edges to nodes in S.

$$(A_G p_S)_{i'} = \sum_{v_j \in S} A_G(v_i, v_j) \frac{1}{|S|} \tag{2.54}$$

$$\leq \frac{\Delta}{2} \frac{1}{\Delta} \frac{1}{|S|} = \frac{1}{2|S|} \tag{2.55}$$

Thus, the corresponding value $(A_G p_S)_i$ of any $v_i \in S$ is *at least* as big as value $(A_G p_S)_{i'}$ of $v_{i'} \notin S$. By a simple greedy argument, we now see that $\sum_{i=1}^{n} \delta_i (A^\ell p_S)_i$ is maximized by picking $\delta_i = 1$ for all nodes in S: To illustrate this, suppose that there is a choice of the δ's such that $\sum_{i=1}^{n} \delta_i (A_G p_s)_i$ is maximized and it holds

$\delta_i < 1$ for some $v_i \in S$. Since no δ can be bigger than 1 and the $\sum_{i=1}^{n} \delta_i = |S|$ there must be a $v_{i'} \notin S$ with $\delta_{i'} > 0$. Since $(A_G p_S)_i \geq (A_G p_S)_{i'}$ decreasing $\delta_{i'}$ and increasing δ_i does not decrease the sum. Thus, choosing $\delta_i = 1$ for all $v_i \in S$ must maximize the term $\sum_{i=1}^{n} \delta_i (A_G p_s)_i$. This yields:

$$\sum_{i=1}^{n} \delta_i (A_G p_S)_i = \sum_{v_i \in S} \sum_{v_j \in S} A_G(v_i, v_j) \frac{1}{|S|} = \frac{1}{|S|} \sum_{v_i \in S} \frac{e(v_i, v_j)}{\Delta} \tag{2.56}$$

$$= \frac{\Delta|S| - O_S}{\Delta|S|} = 1 - \frac{O_S}{\Delta|S|} = 1 - \Phi_G(S) \tag{2.57}$$

Here, the value O_S denotes the edges leaving S. Given that the graph is Δ-regular, the term $\Delta|S| - O_S$ counts all edges in S. This was to be shown. ☐

If we combine our two claims, the lemma follows. ☐

Thus, $\Phi_{G^\ell}(S)$ is *at least as* big as $\Phi_G(S)$. Together with the previous lemmas, this proves Lemma 2.5. ☐

2.2.2 A Cut-Based Union Bound

In this section, we present a variant of the union bound that is necessary for our analysis. Consider a series of *bad* events that do not occur w.h.p., i.e., with probability for these events is smaller than $1/n^c$ for some $c > 1$. If the number of events in question is superpolynomial, i.e., bigger than n^c for any constant c, the union bound alone is not enough to show that these events do not happen w.h.p. In this chapter, we need to make some statements about events that correlate to all possible subsets of nodes of a random graph. Since there are exponentially many of these subsets, we need to apply the union bound more carefully. We show the following technical lemma:

Lemma 2.12 (Cut-based Union Bound). *Let $G := (V, E)$ be a (multi-)graph with n nodes and m edges and let $\mathcal{B} := \{\mathcal{B}_S \mid S \subseteq V \wedge |S| \leq n/2\}$ a set of bad events. Suppose that the following three properties hold:*

1. *G has at most $m \in O(n^{c_1})$ edges for some constant $c_1 > 1$.*
2. *For each $\mathcal{B}_S$ it holds $\mathbf{Pr}[\mathcal{B}_S] \leq e^{-c_2 O_S}$ for some constant $c_2 > 0$.*
3. *The minimum cut of G is at least $\Lambda := 4\frac{c_1}{c_2} c_3 \log n$ edges for some tunable constant $c_3 > 1$.*

Then, the probability any of the events in $\mathcal{B}$ happens can be bounded by:

$$\mathbf{Pr}\left[\bigcup_{S \subset V} \mathcal{B}_S\right] \leq n^{-c_3} \tag{2.58}$$

In other words, w.h.p. no event from $\mathcal{B}$ occurs.

Proof. The core of this lemma is a celebrated result of Karger [Kar00] that bounds the number of cuts (and therefore the number of subsets $S \subset V$ with $|S| \leq n/2$ as one side of each cut must have fewer than $n/2$ nodes) with at most $\alpha\Lambda$ outgoing edges by $O(n^{2\alpha})$. More precisely, it holds:

Lemma 2.13. *Theorem 3.3 in [Karoo], simplified Let G be an undirected, unweighted (multi-)graph and let $\Lambda > 1$ be the size of a minimum cut in G. For an even parameter $\alpha \geq 1$, the number of cuts with at most $\alpha\Lambda$ edges is bounded by $n^{2\alpha}$.*

Thus, if the probability of a bad event for a set S exponentially depends on O_S (and not some constant c), a careful application of the union bound will give us the desired result. The idea behind the proof is to divide all subsets into groups based on the number of their outgoing edges. Then, we use Karger's Theorem to bound the number of sets in a group and use the union bound for each group individually.

More precisely, let $\mathsf{Pow}(V)$ denote all possible subsets of V. Then, we define $\mathcal{S}_\alpha \in \mathsf{Pow}(V)$ to be the set of all sets that have a cut of size $c \in [\alpha\Lambda, 2\alpha\Lambda)$. Using this definition, we can show that the following holds by using the union bound and regrouping the sum:

$$\mathbf{Pr}[\mathcal{B}] \leq \sum_{S \subset V} \mathbf{Pr}[\mathcal{B}_S] \leq \sum_{\alpha=1}^{\frac{m}{\Lambda}} \sum_{S \in \mathcal{S}_\alpha} \mathbf{Pr}[\mathcal{B}_S \mid S \in \mathcal{S}_\alpha] \tag{2.59}$$

Note that the upper limit $\frac{m}{\Lambda}$ of the outermost sum is derived from the fact that at most m edges may cross any cut in the graph. Further note that that for each $\alpha > 1$ it holds that $\mathcal{S}_\alpha \subset \{S \subset V \mid O_S \leq 2\alpha\Lambda\}$ by definition and therefore $|\mathcal{S}_\alpha| \leq |\{S \subset V \mid O_S \leq 2\alpha\Lambda\}|$ Now we can apply Theorem 2.13 and see that

$$\begin{aligned}
\mathbf{Pr}[\mathcal{B}] &\leq \sum_{\alpha=1}^{\frac{m}{\Lambda}} \sum_{S \subset V, O_S \leq 2\alpha\Lambda} \mathbf{Pr}[\mathcal{B}_S \mid S \in \mathcal{S}_\alpha] && (2.60)\\
&\leq \sum_{\alpha=1}^{\frac{m}{\Lambda}} n^{2\cdot(2\alpha)} \max_{S \subset V, O_S \leq 2\alpha\Lambda} \mathbf{Pr}[\mathcal{B}_S \mid S \in \mathcal{S}_\alpha] && (2.61)\\
&\leq \sum_{\alpha=1}^{\frac{m}{\Lambda}} n^{2\cdot(2\alpha)} \max_{S \subset V, O_S \leq 2\alpha\Lambda} \mathbf{Pr}[\mathcal{B}_S \mid O_S \geq \alpha\Lambda] && (2.62)\\
&\leq \sum_{\alpha=1}^{\frac{m}{\Lambda}} n^{2\cdot(2\alpha)} e^{-c_2\Lambda\alpha} && (2.63)
\end{aligned}$$

Here, inequality (2.61) followed from Theorem 2.13, everything else from the definition of S_α and $\mathcal{B}_S$. Finally, our specific choice of $\Lambda \geq 4\frac{c_1}{c_2}c_3 \log n$ comes into play. Plugging it into the exponent of our bound, we get:

$$\mathbf{Pr}[\mathcal{B}] \leq \sum_{\alpha=1}^{\frac{m}{\Lambda}} n^{2\cdot(2\alpha)} n^{-4c_1\cdot c_3\alpha} \leq \sum_{\alpha=1}^{\frac{m}{\Lambda}} n^{-c_1c_3} \leq \frac{m}{\Lambda} n^{-c_1c_3} \tag{2.64}$$

To complete the proof, we need to bound $\frac{m}{\Lambda}$. Per definition, it holds that $m \leq n^{c_1}$. Back in the formula, we get

$$\mathbf{Pr}[\mathcal{B}] \leq \frac{m}{\Lambda} n^{-c_1c_3} \leq \frac{n^{c_1}}{\Lambda} n^{-c_1c_3} \leq \frac{1}{\Lambda} n^{c_3} < n^{-c_3} \tag{2.65}$$

This proves the lemma. □

2.3 Analysis of CreateExpander

The main challenge of our analysis is to show that after $L \in O(\log n)$ iterations, the final graph G_L has a diameter of $O(\log n)$. To show this, we will conduct an induction over the graphs $G_1, \ldots, G_L$. Our main insight is that — given the communication graph is benign — we can use short random walks of *constant* length to iteratively increase the graph's conductance until we reach a graph of low diameter. In the following, we will abuse notation and refer to the virtual nodes V' used in this stage of the algorithm as V. We show the following:

Lemma 2.14. *Let $G_0 := (V, E_0)$ a benign (multi-)graph with n nodes and $O(n^3)$ edges. Let $L := 3\log n$ and $\Lambda := 6400\lambda \log n$ for some tunable $\lambda > 1$. Then, it holds:*

$$\mathbf{Pr}\left[\Phi(G_L) \geq \frac{1}{32}\right] \geq \left(1 - n^{-\lambda}\right) \tag{2.66}$$

Intuitively, this makes sense as the conductance is a graph property that measures how well-connected a graph is, and — since the random walks monotonically converge to the uniform distribution — the newly sampled edges can only increase the graph's connectivity. Our formal proof is structured into four steps: First, in Section 2.3.1, we show that w.h.p. each node receives and sends at most $O(\Delta)$ messages in each round. Thus, no messages are dropped during the algorithm's execution. With this technicality out of the way, we show in Section 2.3.2 that the conductance of G_{i+1} increases by a factor of $\Omega(\sqrt{\ell})$ w.h.p. if G_i is benign. Furthermore, in Section 2.3.3 we show that each G_{i+1} is benign if G_i is benign, w.h.p. Finally, we use the union bound to tie these facts together and prove Theorem 1 in Section 2.3.4.

2.3.1 Bounding the Communication Complexity

Before we go into the proof's more intricate details, let us first prove that all messages are successfully sent during the algorithm's execution. Remember that we assume the (virtual) nodes have a communication capacity of $O(\log n)$. Thus, a node can only send and receive $O(\log n)$ messages as excess messages are dropped arbitrarily. To prove that no message is dropped, we must show that no node receives more than $O(\log n)$ random walk tokens in a single step. However, this is a well-known fact about the distribution of random walks:

Lemma 2.15 (Also shown in [DGS16, CHFSV19, SMPU13]). *Consider a Δ-regular graph $G_i = (V, E)$ where each node starts $\frac{\Delta}{8}$ independent random walk tokens. For a node $v \in V$ and an integer t, let X_v^t be the random variable that denotes the number of tokens at node v in step t of the random walk. Then, it holds $\mathbf{Pr}\left[X_v^t \geq \frac{3\Delta}{8}\right] \leq e^{-\frac{\Delta}{12}}$.*

The lemma follows from the fact that each node receives $\frac{\Delta}{8}$ tokens in expectation, given that all neighbors received $\frac{\Delta}{8}$ tokens in the previous round. For each node $v \in V$ let X_v^t be the number of token it has in step t. Assume that it holds $\mathbb{E}\left[X_v^t\right] = \frac{\Delta}{8}$ for all $v \in V$. In the first step, for $t = 0$, this holds by assumption. As G_i is regular, each node $w \in N(v) \cup \{v\}$ sends $X_w^t \cdot \frac{e(v,w)}{\Delta}$ tokens to v on expectation where $e(v, w)$ counts the number of edges between v and w. This follows from Lemma 2.1 for $\ell = 1$. As each node has Δ incoming edges the expected number of tokens is received by v is:

$$\mathbb{E}\left[X_v^{t+1}\right] = \sum_{w \in N(v) \cup \{v\}} \mathbb{E}\left[X_w^t\right] \cdot \frac{e(w,v)}{\Delta} = \sum_{w \in N(v) \cup \{v\}} \frac{\Delta}{8} \cdot \frac{e(w,v)}{\Delta} \tag{2.67}$$

$$= \sum_{w \in N(v) \cup \{v\}} \frac{e(w,v)}{8} = \frac{1}{8} \cdot \sum_{w \in N(v) \cup \{v\}} e(w,v) = \frac{\Delta}{8}. \tag{2.68}$$

Since all nodes again start with $\frac{\Delta}{8}$ tokens on expectation in step $t+1$, the lemma follows inductively. This proves that for all t and v, it holds:

$$\mathbb{E}\left[X_v^t\right] = \frac{\Delta}{8}. \tag{2.69}$$

For the tail estimate, recall that all walks/tokens are independent. In other words, for each t and each v, the variable X_v^t is the sum of binary independent random variables. Thus, a simple application of the Chernoff bound with $\delta = 2$ (cf. Lemma 1.6) yields the result as

$$\mathbf{Pr}\left[X_v^t \geq \frac{3\Delta}{8}\right] = \mathbf{Pr}\left[X_v^t \geq (1+2)\frac{\Delta}{8}\right] = \mathbf{Pr}\left[X_v^t \geq (1+2)\mathbb{E}\left[X_v^t\right]\right] \leq e^{-\frac{2}{3}\frac{\Delta}{8}} = e^{-\frac{\Delta}{12}} \tag{2.70}$$

Note that this Lemma also directly implies that, w.h.p., all random walks create an edge as every possible end-point receives less than $\frac{3\Delta}{8}$ token and therefore replies to all of them. For our concrete value of Δ it holds:

Lemma 2.16. *Let $\Delta \geq \Lambda := 6400\lambda \log n$ for some tunable parameter $\lambda > 1$. Then, for any round t it holds with probability at least $1 - n^{-8\lambda}$ that every node holds fewer than $\frac{3}{8}\Delta$ tokens.*

Proof. This follows directly from Lemma 2.15. Denote X_v^t the number of tokens that node $v \in V$ receives after t steps. Consider the event that node v receives more than $\frac{3}{8}\Delta$ tokens. Recall that the probability for this event is

$$\mathbf{Pr}\left[X_v^t \geq \frac{3}{8}\Delta\right] \leq e^{-\frac{\Delta}{12}} \qquad \triangleright \textit{By Lemma 2.15} \tag{2.71}$$

Now we use that Δ is at least as big as $\Lambda := 6400\lambda \log n$. Plugging this into the formula yields:

$$\mathbf{Pr}\left[X_v^t \geq \frac{3}{8}\Delta\right] \leq e^{-\frac{\Delta}{12}} \leq e^{-9\lambda \log n} = n^{-9\lambda} \tag{2.72}$$

Finally, let $\mathcal{B}$ the event that *any* node receives more than $\frac{3}{8}\Delta$ tokens. By the union bound, we see

$$\mathbf{Pr}[\mathcal{B}] = \mathbf{Pr}\left[\bigcup_{v \in V} X(v,\ell) \geq \frac{3}{8}\Delta\right] \leq \sum_{v \in V} \mathbf{Pr}\left[X(v,\ell) \geq \frac{3}{8}\Delta\right] \qquad \triangleright \textit{Union bound} \tag{2.73}$$

$$\leq \sum_{v \in V} n^{-9\lambda} \leq n^{-8\lambda} \qquad \triangleright \textit{By Equation (2.72)} \tag{2.74}$$

□

Therefore, all nodes receive less than $\frac{3\Delta}{8}$ tokens each round and the algorithm stays within the congestion bounds of our model with probability $1 - n^{-8\lambda}$. Since all iterations take $\ell \cdot L \in O(\log n)$ rounds in total, the union bound implies that no node receives too many messages in any round, w.h.p.

2.3.2 Bounding the Conductance of G_i

In this section, we show that the graph's conductance is increasing by a factor $\Omega(\sqrt{\ell})$ from G_i to G_{i+1} w.h.p. if G_i is benign. More formally, we show the following:

Lemma 2.17. *Let $\lambda > 0$ be a parameter and let G_i and G_{i+1} be the graphs created in iteration i and $i+1$, respectively. Finally, assume that G_i is benign with a minimum cut of at least $\Lambda \geq 6400\lambda \log n$ and degree $\Delta > 64\Lambda$. Then, it holds with probability at least $1 - n^{-7\lambda}$ that*

$$\Phi_{G_{i+1}} \geq \min\left\{\frac{1}{32}, \frac{1}{640}\sqrt{\ell}\Phi_{G_i}\right\} \tag{2.75}$$

In particular, for any $\ell \geq (2 \cdot 640)^2$, it holds

$$\Phi_{G_{i+1}} \geq \min\left\{\frac{1}{32}, 2 \cdot \Phi_{G_i}\right\} \tag{2.76}$$

Our first observation is the fact that random walks of length ℓ are distributed according to 1-step walks in G_i^ℓ. In particular, if we consider a subset $S \subset V$ and pick a node $v \in S$ uniformly at random, then $\Phi_{G_i^\ell}(S)$ denotes the probability that a random walk started at v ends outside of the subset after ℓ steps.

Lemma 2.18. *Let G be a Δ-regular graph and $S \subset V$ be a any subset of nodes with $|S| \leq \frac{n}{2}$ and suppose each node in S starts $\frac{\Delta}{8}$ random walks. Let $\mathcal{Y}_S$ count the ℓ-step random walks that start at some node in $v \in S$ and end at some node $w \in V \setminus S$. Then, it holds:*

$$\mathbb{E}\left[\mathcal{Y}_S\right] := \frac{\Delta|S|}{8}\Phi_{G_i^\ell}(S)$$

Proof. First, we observe that we can express $\mathcal{Y}_S$ as the sum of binary random variables for each walk. For each $v_i \in S$ let $Y_i^1, \ldots, Y_i^d$ be indicator variables that denote if a token started by v_i ended in $\overline{S} := V \setminus S$ after ℓ steps. Given this definition, we see that

$$\mathcal{Y}_S := \sum_{i=1}^{|S|}\sum_{j=1}^{\frac{\Delta}{8}} Y_i^j. \tag{2.77}$$

Recall that an ℓ-step random walk in G_i corresponds to a 1-step random walk in G_i^ℓ. This means that for each of its $\frac{\Delta}{8}$ tokens node v_j picks one of its outgoing edges in G_i^ℓ uniformly at random and sends the token along this edge (which corresponds to an ℓ-step walk). For ease of notation, let O_j^ℓ be the number of edges of node $v_j \in S$ in G_i^ℓ where the other endpoint is not in S, i.e.,

$$O_j^\ell := \sum_{w \in \overline{S}} P_\ell(v_j, w) \tag{2.78}$$

Now consider the k^{th} random walk started at v_j and observe Y_j^k. Note that it holds:

$$\begin{aligned}
\mathbb{E}[Y_k^j(t)] &= \sum_{w \in \overline{S}} \mathbf{Pr}\Big[X_{v_j}^1(w, G^\ell)\Big] \cdot \mathbb{E}[Y_k^j(t) \mid X_{v_j}^1(w, G^\ell)] && (2.79)\\
&= \sum_{w \in \overline{S}} \mathbf{Pr}\Big[X_{v_j}^1(w, G^\ell)\Big] = \sum_{w \in \overline{S}} \frac{P_\ell(v_j, w)}{\Delta^\ell} && (2.80)\\
&= \frac{O_j^\ell}{\Delta^\ell} && (2.81)
\end{aligned}$$

Here, equation (2.80) follows from Lemma 2.1. Let O_S^ℓ be the number of all outgoing edges from the whole set S in G_i^ℓ. It holds that $O_S^\ell := \sum_{v_j \in S} O_j^\ell$. Recall that the definition of $\Phi_{G_i^\ell}$ is the ratio of edges leading out of S and all edges with at least one endpoint in S. Given that G_i^ℓ is a Δ^ℓ-regular graph, a simple calculation yields:

$$\begin{aligned}
\mathbb{E}\left[\mathcal{Y}_S\right] &= \mathbb{E}\left[\sum_{j=1}^{|S|} \sum_{k=1}^{\frac{\Delta}{8}} Y_j^k\right] = \sum_{j=1}^{|S|} \sum_{k=1}^{\frac{\Delta}{8}} \mathbb{E}\left[Y_j^k\right] && (2.82)\\
&= \frac{\Delta}{8} \frac{\sum_{i=1}^{|S|} O_i^\ell}{\Delta^\ell} = \frac{\Delta}{8} \frac{O_S^\ell}{\Delta^\ell} && \triangleright \textit{By Eq. (2.81)} \quad (2.83)\\
&= \frac{\Delta|S|}{8} \frac{O_S^\ell}{|S|\Delta^\ell} = \frac{\Delta|S|}{8} \Phi_{G_i^\ell}(S) && (2.84)
\end{aligned}$$

In the last line , we used that $\Phi_{G_i^\ell}(S) := \frac{O_S^\ell}{|S|\Delta^\ell}$ per definition as O_S^ℓ counts the edges with an endpoint outside of S and $|S|\Delta^\ell$ counts the total number edges with an endpoint in S as G_i^ℓ is a Δ^ℓ-regular graph. This proves the lemma. $\square$

Therefore, a lower bound on $\Phi_{G_i^\ell}$ gives us a lower bound on the expected number of tokens that leave any set S. Thus, as long as G_i is regular and lazy, we have a suitable lower bound for $\Phi_{G_i^\ell}$ with Lemma 2.5. In fact, given that the random walks are independent, we can even show the following:

Lemma 2.19. *Let $S \subset V$ be set of nodes with O_S outgoing edges, then it holds:*

$$\mathbf{Pr}\left[\mathcal{Y}_S \leq \frac{\Delta|S|}{16} \Phi_{G_i^\ell}(S)\right] \leq e^{-\frac{O_S}{64}} \tag{2.85}$$

Proof. This follows from the Chernoff bound and the fact that the random walks are independent. Recall that for each set S, the number of outgoing edges $\mathcal{Y}_S := \sum_{i=1}^{s}\sum_{k=1}^{\Delta/8} Y_i^k$ in G_{i+1} is determined by a series independent binary variables. Thus, by the Chernoff bound, it holds that

$$\mathbf{Pr}[\mathcal{Y} \leq (1-\delta)\mathbb{E}[\mathcal{Y}]] \leq e^{-\frac{\delta^2}{2}\mathbb{E}[\mathcal{Y}]}. \tag{2.86}$$

By choosing $\delta = 1/2$, we get

$$\mathbf{Pr}\left[\mathcal{Y} \leq \frac{1}{2}\mathbb{E}\left[\mathcal{Y}_S\right]\right] \leq e^{-\frac{\mathbb{E}[\mathcal{Y}_S]}{8}}. \tag{2.87}$$

Therefore, it remains to show that our claim that $\mathbb{E}[\mathcal{Y}] \geq \frac{O_S}{8}$ holds, and we are done. By Lemma 2.18 we have for all set with O_S outgoing edges that

$$\mathbb{E}[\mathcal{Y}_S] \geq \frac{\Delta|S|}{8}\Phi_{G_i^\ell}(S) \qquad \triangleright \textit{By Lemma 2.18} \tag{2.88}$$

$$\geq \frac{\Delta|S|}{8}\Phi_{G_i}(S) \qquad \triangleright \textit{By Lemma 2.10} \tag{2.89}$$

$$\geq \frac{\Delta|S|}{8}\frac{O_S}{\Delta|S|} \geq \frac{O_S}{8} \qquad \triangleright \textit{As } \Phi(S) := \frac{O_S}{\Delta|S|} \tag{2.90}$$

This proves the lemma. □

Recall that we need to show that *every* subset $S \subset V$ with $|S| \leq n/2$ has a conductance of $O(\sqrt{\ell}\Phi_{G_i})$ in G_{i+1} in order to prove that $\Phi_{G_{i+1}} = \Omega(\sqrt{\ell}\Phi_{G_i})$. In the following, we want to prove that for every set S, there are at least $\Theta(\sqrt{\ell}\Phi_{G_i})$ tokens that start at some node $v \in S$ and end at some node $w \in \overline{S}$ after ℓ steps w.h.p. These tokens are counted by the random variable $\mathcal{Y}_S$, which we analyzed above. In particular, given that a set S has O_S outgoing edges, the value of $\mathcal{Y}_S$ is concentrated around its expectation with probability $e^{-\Omega(O_S)}$. Thus, we can apply Lemma 2.12 and show that — for a big enough Λ — all sets have many tokens that escape.

Lemma 2.20. *Define* $\mathcal{M}(S,\ell) := \min\left\{\frac{\Delta|S|}{32}, \frac{\Delta|S|}{640}\sqrt{\ell}\Phi_{G_i}\right\}$. *Let* G_i *be a benign graph with* $\Lambda \geq 6400\lambda\log n$. *Then, it holds*

$$\mathbf{Pr}[\forall S \subset V : \mathcal{Y}_S \geq \mathcal{M}(S,\ell)] \geq 1 - n^{-8\lambda}. \tag{2.91}$$

Proof. For a set $S \subset V$ we define $\mathcal{B}_S$ to be the event that S has *bad conductance*, i.e., it holds that $\mathcal{Y}_S$ — the number of tokens that leave S — is smaller than $\mathcal{M}(S,\ell)$. We let $\mathcal{B}_1 = \bigcup_{S\subset V}\mathcal{B}_S$ be the event that there exists a set S with bad conductance, i.e., there is any $\mathcal{B}_S$ that is true. By Lemma 2.19, we know that the probability of $\mathcal{B}_S$ exponentially depends on O_S. Thus, we want to use Lemma 2.12 to that no $\mathcal{B}_S$ occurs w.h.p. Therefore, we must show that the three conditions mentioned in the lemma are fulfilled. The first and third property follow directly from the definition of benign graphs, as the graph is polynomial in size and has a logarithmic minimum cut with a tunable constant. For the concrete constants, it holds:

1. G **has at most** $m \in O(n^{c_1})$ **edges for some constant** $c_1 > 1$**:** Recall that we limited ourselves to simple initial graphs with $O(n^2)$ edges and copied each edge $O(\Lambda)$ times. Since $\Lambda \in o(n)$, we have strictly less than n^3 edges. Thus, we have $c_1 := 3$.

2. **For each $\mathcal{B}_S$ it holds Pr$[\mathcal{B}_S] \leq e^{-c_2 O_S}$ for some constant $c_2 > 0$:** By Lemma 2.19 a bad event $\mathcal{B}_S$ for a set $S \subseteq V$ happens with probability at most $e^{-\frac{O_S}{64}}$. Thus, we have $c_2 := \frac{1}{64}$.

3. **The minimum cut of G is at least $\Lambda := 4\frac{c_1}{c_2}c_3 \log n$ edges for some tunable variable $c_3 > 1$:** Since G_i is benign, it holds $\Lambda \geq 6400\lambda \log n > (4 \cdot 3 \cdot 64)\, 8\lambda \log n$ for a constant λ. Thus, we have $c_3 := 8\lambda$. Note that λ is tunable as it can be chosen as high as we want by constructing a sufficiently large minimum cut in G_0 by creating copies of each initial edge.

Given that all three conditions are fulfilled with constants $c_1 = 3$,$c_2 = \frac{1}{64}$, and $c_3 = 8\lambda$, Lemma 2.12 implies that it holds:

$$\mathbf{Pr}\left[\bigcup_{S \subseteq V} \mathcal{B}_S\right] \leq n^{-c_3} := n^{-8\lambda} \tag{2.92}$$

This was to be shown. □

With this insight, we can now formally prove Lemma 2.17:

Proof of Lemma 2.17. First, we note that if no set $S \subset V$ with $|S| \leq n/2$ has less than $\mathcal{M}(S,\ell)$ tokens that end outside of S and all tokens are used to create an edge, then the resulting conductance of G_{i+1} must also be at least $\mathcal{M}(S,\ell)$ and the lemma follows. This can easily be verified by observing the algorithm: We note that by construction, the degree can *never* be higher than Δ. Recall that every node creates its edges for G_{i+1} based on the tokens it received. If any node receives fewer than Δ tokens, it creates self-loops to reach a degree of Δ. Excess edges are dropped arbitrarily to ensure a degree of at most Δ. Thus, each set S maintains $\Delta|S|$ edges in total, as each node will always have Δ edges regardless of how many tokens it receives. Further, we let $O_S^{(i+1)}$ be the number of edges that leave S in G_{i+1}. Two sets of edges determine the value of $O_S^{(i+1)}$:

1. The edges that are created by nodes outside S. These are based on tokens they received from nodes in S. We denote these edges by $\widetilde{\mathcal{Y}}_S$.

2. The edges that are created by nodes in S. These are based on tokens they received from nodes outside. We do not make any statements about these edges and ignore them for the remainder of the proof.

For easier notation, let $\overline{\mathcal{B}_1}$ be the event that each set $S \subset V$ has $\mathcal{M}(S,\ell)$ tokens that end outside of the set. Moreover, let $\mathcal{B}_1$ be the event that this is not the case. Further, let $\overline{\mathcal{B}_2}$ be the event that all these tokens are used to create an edge. Again, let $\overline{\mathcal{B}_2}$ denote the complementary event. Note that the event $\overline{\mathcal{B}}_2$ directly implies that $\widetilde{\mathcal{Y}}_S = \mathcal{Y}_S$. Given these facts and definitions, we can use Lemma 2.18 from above and see that the events $\overline{\mathcal{B}_1}$ and

$\overline{\mathcal{B}_2}$ imply that the conductance of each set is bigger or equal to $\mathcal{M}(S, \ell)$. If they both occur simultaneously, it holds for each set $S \subset V$:

$$\overline{\mathcal{B}_1} \cap \overline{\mathcal{B}_2} \Rightarrow \left\{ \mathcal{Y}_S \geq \min\left\{ \frac{\Delta|S|}{32}, \frac{\Delta|S|\sqrt{\ell}\Phi_{G_i}}{640} \right\} \right\} \cap \{\widetilde{\mathcal{Y}_S} = \mathcal{Y}_S\} \tag{2.93}$$

$$\Rightarrow \left\{ \widetilde{\mathcal{Y}}_S \geq \min\left\{ \frac{\Delta|S|}{32}, \frac{\Delta|S|\sqrt{\ell}\Phi_{G_i}}{640} \right\} \right\} \tag{2.94}$$

$$\Rightarrow \left\{ O_S^{(i+1)} \geq \min\left\{ \frac{\Delta|S|}{32}, \frac{\Delta|S|\sqrt{\ell}\Phi_{G_i}}{640} \right\} \right\} \tag{2.95}$$

$$\Rightarrow \left\{ \frac{O_S^{(i+1)}}{\Delta|S|} \geq \min\left\{ \frac{1}{32}, \frac{\sqrt{\ell}\Phi_{G_i}}{640} \right\} \right\} \tag{2.96}$$

$$\Rightarrow \left\{ \Phi_{G_{i+1}}(S) \geq \min\left\{ \frac{1}{32}, \frac{\sqrt{\ell}\Phi_{G_i}}{640} \right\} \right\} \tag{2.97}$$

Since this implication holds for all sets $S \subset V$, we get:

$$\overline{\mathcal{B}_1} \cap \overline{\mathcal{B}_2} \Rightarrow \left\{ \Phi_{G_{i+1}} \geq \min\left\{ \frac{1}{32}, \frac{\sqrt{\ell}\Phi_{G_i}}{640} \right\} \right\} \tag{2.98}$$

Therefore, it holds that

$$\begin{aligned}
\mathbf{Pr}\left[\Phi_{G_{i+1}} \geq \frac{\sqrt{\ell}\Phi_{G_i}}{640}\right] &\geq \mathbf{Pr}\left[\overline{\mathcal{B}_1} \cap \overline{\mathcal{B}_2}\right] && \rhd \textit{By Equations (2.98)} && (2.99)\\
&\geq 1 - (\mathbf{Pr}[\mathcal{B}_1] + \mathbf{Pr}[\mathcal{B}_2]) && \rhd \textit{Union bound} && (2.100)\\
&\geq 1 - \left(n^{-8\lambda} + n^{-8\lambda}\right) && \rhd \textit{By Lemmas 2.20 and 2.16} && (2.101)\\
&\geq 1 - n^{-7\lambda} && && (2.102)
\end{aligned}$$

This proves the lemma. □

2.3.3 Ensuring That Each G_i is Benign

Since all the arguments from before only hold if G_i is benign, we must make sure that each graph in $\mathcal{G} := G_1, \ldots, G_L$ is indeed benign. As before, we prove this step by step and show that G_{i+1} is benign given that G_i is benign.

Lemma 2.21. *Let G_i and G_{i+1} be the graphs created in iteration i and $i+1$, respectively, and assume that G_i is benign. Then with probability at least $1 - n^{-8\lambda}$ the graph G_{i+1} is also benign*

Proof. We will show that each G_{i+1} is a Δ-regular, lazy graph with a Λ-sized cut. Note that the last property also ensures that G_{i+1} is connected. The first property follows directly from observing the algorithm: Recall that every node creates its edges for G_{i+1} based on the tokens it received. If any node receives fewer than Δ tokens, it creates self-loops to reach a degree of Δ. Excess edges are dropped arbitrarily to ensure a degree of at most Δ. By a similar argument, we can easily see that G_{i+1} lazy. For this, recall that a node connects to endpoints of all its $\frac{\Delta}{8}$ tokens and additionally to the origins of all (but at most $\frac{3\Delta}{8}$) tokens it received. Thus, in the worst case, it creates at most $\frac{\Delta}{8} + \frac{3\Delta}{8} = \frac{\Delta}{2}$ edges. Therefore, it creates at least $\frac{\Delta}{2}$ — and therefore enough — self-loops. The third property — the Λ-sized minimum cut — is perhaps the most difficult to show. However, at a closer look, the proof is almost identical to the proof of Lemma 2.17. In particular, we show that all cuts that are *close* to the minimum cut will (in expectation and w.h.p.) increase in size in each iteration and never fall below Λ. The idea behind the proof uses the fact that [KL14] (and therefore Lemma 2.5) gives us a stronger bound on the expected growth of the subset than just the conductance. This observation is enough to show that all sets that have close to Λ outgoing connections will slightly increase the number of outgoing connections for a big enough ℓ. In particular, it holds:

Lemma 2.22. *Suppose that $\ell \geq (2 \cdot 640)^2$. Then, for any set $S \subseteq V$ with $|S| \leq \frac{n}{2}$ and O_S outgoing edges, it holds:*

$$\mathbf{Pr}[\mathcal{Y}_S \leq \Lambda] \leq e^{-\frac{O_S}{64}} \tag{2.103}$$

Proof. By Lemma 2.19 we have that:

$$\mathbf{Pr}\left[\mathcal{Y} \leq \frac{1}{2}\left(\frac{\Delta|S|}{8}\Phi_{G_i^\ell}(S)\right)\right] \leq e^{\frac{O_S}{64}}. \tag{2.104}$$

Thus, it remains to show that for all sets $S \subseteq V$, it holds:

$$\frac{\Delta|S|}{8}\Phi_{G_i^\ell}(S) \leq 2\Lambda \tag{2.105}$$

Consider a set of size $|S| := \delta_s n$. For this set, it holds:

$$\mathbb{E}[\mathcal{Y}_S] = \frac{\Delta|S|}{8}\Phi_{G_i^\ell}(S) \geq \frac{\Delta|S|}{8}\Phi_\delta(G_i^\ell) \tag{2.106}$$

This follows because $\Phi_{\delta_s}(G_i^\ell) \leq \Phi_{G_i^\ell}(S)$ per definition. Due to Lemma 2.5, we know that we can bound this as follows:

Case 1: $\delta_s \geq \frac{1}{4}$ It holds:

$$\Phi_{G^\ell,\delta_s} \geq \min\left\{\frac{1}{2}, \frac{1}{40}\sqrt{\ell}\Phi(G)\right\} \qquad \triangleright \textit{By Lemma 2.5} \tag{2.107}$$

$$\geq \min\left\{\frac{1}{2}, \frac{1}{40}\sqrt{\ell}\frac{2\Lambda}{\Delta n}\right\} \qquad \triangleright \textit{By Lemma 2.2} \tag{2.108}$$

$$\geq \min\left\{\frac{1}{2}, \frac{1}{40}\sqrt{\ell}\frac{\Lambda}{2\Delta|S|}\right\} \qquad \triangleright \textit{As } |S| \geq n/4 \tag{2.109}$$

Case 2: $\delta_s \leq \frac{1}{4}$ It holds:

$$\Phi_{G^\ell,2\delta_s} \geq \min\left\{\frac{1}{4}, \frac{1}{40}\sqrt{\ell}\Phi_{2\delta_s}\right\} \qquad \rhd \textit{By Lemma 2.5} \tag{2.110}$$

$$\geq \min\left\{\frac{1}{4}, \frac{1}{40}\sqrt{\ell}\frac{2\Lambda}{\Delta 2\delta_s n}\right\} \qquad \rhd \textit{By Lemma 2.2} \tag{2.111}$$

$$\geq \min\left\{\frac{1}{4}, \frac{1}{40}\sqrt{\ell}\frac{\Lambda}{\Delta|S|}\right\} \qquad \rhd \textit{As } |S| := \delta_s n \tag{2.112}$$

The factor of 2 that appears in the denominator in the second line results from a subtle detail in Lemma 2.5. Since we observe a set of size $\delta_s n$, we must consider $\Phi_{2\delta_s}$. Since we always consider sets of size at most $\frac{n}{4}$, this is always well-defined.

Putting these bounds back into the formula gives us (for a set of any size):

$$\mathbb{E}[\mathcal{Y}_S] \geq \frac{\Delta|S|}{8} \min\left\{\frac{1}{4}, \frac{1}{40}\sqrt{\ell}\frac{\Lambda}{2\Delta|S|}\right\} \tag{2.113}$$

Now, choosing $\ell > (2 \cdot 640)^2$ yields:

$$\mathbb{E}[\mathcal{Y}_S] \geq \frac{\Delta|S|}{8} \min\left\{\frac{1}{4}, \frac{1}{40}\sqrt{(2\cdot 640)^2}\frac{\Lambda}{2\Delta|S|}\right\} \tag{2.114}$$

$$\geq \frac{\Delta|S|}{8} \min\left\{\frac{1}{4}, \frac{1}{16}\frac{\Lambda}{\Delta|S|}\right\} \tag{2.115}$$

We need to again distinguish between two cases:

Case 1 : $\frac{1}{4} \leq \frac{1}{16}\frac{\Lambda}{\Delta|S|}$ In this case, we have:

$$\mathbb{E}[\mathcal{Y}_S] \geq \frac{\Delta|S|}{8}\frac{1}{4} \geq \frac{\Delta|S|}{32} \geq \frac{64\Lambda}{32} = 2\Lambda \tag{2.116}$$

Here, we used that $\Delta \geq 64\Lambda$ as G_i is benign.

Case 2 : $\frac{1}{4} > \frac{1}{16}\frac{\Lambda}{\Delta|S|}$ In this case, we have:

$$\mathbb{E}[\mathcal{Y}_S] \geq \frac{\Delta|S|}{8}\frac{16\Lambda}{\Delta|S|} = 2\Lambda \tag{2.117}$$

This proves that $\mathbb{E}[\mathcal{Y}_S] \leq 2\Lambda$ and therefore the lemma. □

We can round up the proof with the same trick as before. Again, we must show that *every* cut has a value of at Λ and use Karger's bound together with Lemma 2.22 to show that no cut has a worse value, w.h.p.

Lemma 2.23. *Let G_i be a benign graph with $\Lambda \geq 6400\lambda\log n$. Then, it holds*

$$\mathbf{Pr}[\forall S \subset V : \mathcal{Y}_S \geq \Lambda\}] \geq 1 - n^{-8\lambda}. \tag{2.118}$$

Proof. The proof is completely analogous to the proof of Lemma 2.20 down to the constants. For each set S with $|S| \leq n/2$, we observe the event $\mathcal{B}_S$ that $\mathcal{Y}_S$ is smaller than Λ. Just as in Lemma 2.20, the graph G_i has at most n^3 edges, the event $\mathcal{B}_S$ has probability at most $e^{-\frac{O_S}{64}}$, and the minimum cut is of size $6400\lambda \log n$. Thus, Lemma 2.12 yields this lemma. $\square$

Finally, recall that each token creates an edge with probability $1 - n^{-8\lambda}$ by Lemma 2.19 and at least Λ tokens leave each set with prob. $1 - n^{-8\lambda}$ by Lemma 2.23. By the union bound, both these events hold with prob. at least $1 - n^{-7\lambda}$. This implies that each cut in G_{i+1} has at least size Λ w.h.p. and therefore proves Lemma 2.21

$\square$

2.3.4 Finalizing the Proof

To round up the analysis, we only need to prove Lemma 2.14 and show that after $O(\log n)$ iterations, the graph has constant conductance. Based on our insights, we can conclude that if Δ, ℓ, and Λ are big enough and G_i is benign, then G_{i+1} is benign and has at least twice its conductance w.h.p. (if it was not already constant). To be precise, assume that G_i is benign and let $\ell := (2 \cdot 640)^2$ and $\Lambda \geq 6400\lambda \log n$. Then, it holds that with probability $1 - n^{-7\lambda}$ that

1. G_{i+1} has conductance at least $2\Phi_{G_i}$ (if Φ was not already constant) by Lemma 2.17, and
2. G_{i+1} is benign by Lemma 2.21.

For an easier notion, let $\mathcal{Z}_{i+1}$ be a random variable that takes the value of G_{i+1}'s conductance if G_{i+1} benign and 0 otherwise. For any value $\varphi > (0, \frac{1}{32})$, it holds that:

$$\mathbf{Pr}[\mathcal{Z}_{i+1} \geq \varphi] \geq \mathbf{Pr}\left[\mathcal{Z}_i \geq \frac{\varphi}{2}\right]\mathbf{Pr}\left[\mathcal{Z}_{i+1} \geq \varphi \mid \mathcal{Z}_i \geq \frac{\varphi}{2}\right] \tag{2.119}$$

$$\geq \mathbf{Pr}\left[\mathcal{Z}_i \geq \frac{\varphi}{2}\right]\left(1 - (2n^{-7\lambda})\right) \tag{2.120}$$

Here, the first inequality follows from the law of total expectation and the last follows from Lemmas 2.17 and 2.21 as well as the union bound. Given that after every iteration the graph's conductance increases by a factor of 2 w.h.p, Inequality (2.120) implies the following:

$$\mathbf{Pr}\left[\mathcal{Z}_{i+1} \geq \frac{2^{i+1}}{2^L 32}\right] \geq \mathbf{Pr}\left[\mathcal{Z}_i \geq \frac{2^i}{2^L 32}\right]\left(1 - (2n^{-7\lambda})\right) \tag{2.121}$$

If we inductively apply this argument L times, we get:

$$\mathbf{Pr}\left[\mathcal{Z}_L \geq \frac{1}{32}\right] \geq \mathbf{Pr}\left[\mathcal{Z}_{L-1} \geq \frac{1}{64}\right]\left(1 - (2n^{-7\lambda})\right) \geq \ldots \geq \mathbf{Pr}\left[\mathcal{Z}_0 \geq \frac{1}{2^L 32}\right]\left(1 - (2n^{-7\lambda})\right)^L \tag{2.122}$$

By choosing $L := 3\log n$, we obtain:

$$\mathbf{Pr}\left[\mathcal{Z}_L \geq \frac{1}{32}\right] \geq \mathbf{Pr}\left[\mathcal{Z}_0 \geq \frac{1}{32n^3}\right]\left(1 - n^{-6\lambda}\right) \tag{2.123}$$

Since the minimal conductance of any benign graph is $O(n^{-3})$ by Lemma 2.2, it follows:

$$\mathbf{Pr}\left[\mathcal{Z}_L \geq \frac{1}{32}\right] \geq \left(1 - n^{-6\lambda}\right) \tag{2.124}$$

Therefore, the graph, w.h.p., has a constant conductance after $O(\log n)$ iterations. This takes $O(\log n)$ rounds, since each iteration lasts only $\ell \in O(1)$ rounds. Finally, a constant conductance implies a logarithmic diameter by Lemma 2.3. This concludes the analysis.

3 Fast Computation of Connected Components

This section shows how our algorithm can be extended to find connected components in an arbitrary graph G. In particular, for each connected component C of G, we want to establish a well-formed tree of overlay edges that contains all nodes of C. If the graph is connected, i.e., there is exactly one component with n nodes, this theorem simply translates Theorem 1 to the HYBRID model. The main result of this section is the following theorem:

Theorem 2. *Time-Optimal Construction of Overlays (HYBRID)*

Let $G = (V, E)$ be a bidirected graph with n nodes and suppose that the largest connected component contains n' nodes. Then, there is a randomized algorithm in the HYBRID model that constructs a well-formed tree of each connected component in $O(\log n')$ rounds, w.h.p.

Each node sends at most $\gamma \in O(\log^3 n)$ bits per round in global mode, w.h.p., and $\lambda \in O(\log n)$ bits along each edge in the local mode.

Note that the main difficulty preventing us from applying Theorem 1 directly on each component when we use the HYBRID model is that the initial graph's degree is unbounded. Note that in the HYBRID model, a node $v \in V$ can no longer create $O(\mathtt{deg}(v) \log n)$ overlay edges, which was possible in the P2P-CONGEST model. Thus, high degree nodes cannot simply simulate the algorithm from before. Therefore, the main contribution of this section is a *preprocessing* algorithm that transforms any connected subgraph of G into a graph of bounded degree $O(\log n)$. Then, we execute the algorithm of Theorem 1 to create a well-formed tree for each component. By Theorem 1, this takes $O(\log n')$ time, w.h.p., for parameters $L \in O(\log n')$. Further, because the degree is $O(\log n)$, each node has to send $O(\Lambda \log n)$ messages of size $O(\log n)$ for $\Lambda \in O(\log n)$. Thus, the global capacity required by each node is $O(\log^3 n)$ as claimed. Therefore, we only need to prove the following lemma.

Lemma 3.1. *Let $G = (V, E)$ be a bidirected graph where each connected component contains at most n' nodes. There exists a randomized algorithm that transforms G into a bidirected graph $H := (V, E_H)$ that has degree $O(\log n)$ and in which two nodes lie in the same component if and only if they lie in the same component in G. The algorithm takes $O(\log n')$ rounds w.h.p., in the CONGEST model.*

The algorithm will be executed in parallel on all connected components of the graph G. In the remainder, we will w.l.o.g. focus on a single connected component C and its implied subgraph $G_C := (V_C, E_C)$. In particular, we present an algorithm that creates a bounded degree graph $H(G_C)$for the nodes of G_C, so our main algorithm can transform it into a well-formed tree in $O(\log n')$ time. Since there is no communication between components and we run the algorithm independently for each component, focusing on a single component is enough to prove the lemma.

The algorithm's main idea is to first eliminate *most* edges by constructing a sparse spanner. Note that well-known spanner constructions take $O(\log n)$ time, so a more careful analysis is required to show that we can construct such a graph in $O(\log n')$ time, i.e., logarithmic in the size of the biggest connected component. In particular, we transform the graph into a sparse spanner[1] using the efficient spanner construction of Miller et al. [MPVX15a], later refined by Elkin and Neiman [EN18a]. In their algorithm, each node $v \in V$ draws an exponential random variable δ_u and broadcasts it to all nodes in the graph [2]. Let u_v^* be the node that maximizes $\delta_{u_v^*} - \mathsf{dist}(u_v^*, v)$ where $\mathsf{dist}(u_v^*, v)$ is the hop distance between u_v^* and v. Then, v joins the cluster of u_v^* by storing its identifier and adding the first edge over which it received the broadcast from u_v^* to the spanner. Finally, the nodes share their clusters' identifiers with their neighbors and add edges between clusters to obtain a connected graph. It is known that the resulting subgraph of G has very few edges as each node has only a few neighbors in different clusters. As we will see, this property implies that we will have relatively few nodes of high degree, and all these nodes have many neighbors of low degree. This follows from the fundamental properties of the exponential distribution. We show that if the size of each connected component in graph G is bounded by n', it suffices to observe variables δ_v that are conditioned on being smaller than $4 \log n'$, i.e., truncated exponential random variables. With this small change, we speed up the algorithm to $O(\log n')$ while still producing a sparse subgraph with fewer edges.

Then, in a second step, we let all remaining nodes of high degree *delegate* their edges to nodes of lower degree. This roughly means that each node introduces its neighbor to each other in a way that preserves the connectivity but massively reduces its own degree (if it is higher than $\omega(\log n)$) while only slightly increasing the neighbors' degrees. If every node of a high degree has sufficiently many neighbors of a low degree, the overall degree becomes small enough for our algorithm to handle.

The remainder of this chapter is structured as follows. In Section 3.1 we describe our algorithm and in Section 3.2 we prove Lemma 3.1, which along with Theorem 1 implies Theorem 2.

[1] We remark that — although we use the term spanner — we are not interested in the approximation of shortest paths but only in the sparsification of the initial graph.

[2] We describe the concrete implementation of this broadcast in Chapter 3 and assume for an easier explanation that it is possible for the moment.

3.1 Algorithm Description

In the following, we present the two steps of our preprocessing algorithm, the spanner construction and the delegation of edges in detail.

(Step 1) Create a Sparse Spanner $S(G_C)$: In the first step, we will construct a spanner $S(G_C) := (V_C, S(E_C))$ of G_C to reduce the number of edges to $O(n \log n)$ and its arboricity to $O(\log n)$. In particular, we note that $S(G_C)$ is a subgraph of G, so every edge in $S(G_C)$ is a local edge in our hybrid model. We will adapt spanner construction algorithms based on *exponential start time clustering* [Xu17].

Conceptually, our algorithm can be subdivided into two phases. First, we construct clusters of nodes. Each cluster is a subset of nodes that is internally connected via a spanning tree. Then, in the second phase, we add additional edges to connect the clusters. We begin with the description of *clustering phase* where each node joins some cluster. By *joining a cluster*, we mean that a node $v \in V$ either declares itself the *head* of a cluster or picks a neighbor $p(w) \in N_v$, such that the edge $(v, p(w))$ is the next edge of a path to the head of the cluster. In the latter case, we will refer to $p(w)$ as the *predecessor* of v. The clusters are constructed as follows.

1. In the beginning, each node v independently draws a random value r_v from the exponential distribution with parameter $\beta = 1/2$. Values larger than $4 \log n'$ are redrawn until $r_v \leq 4 \log n'$, i.e., we draw exponentially distributed variables conditioned on being smaller than $4 \log n'$.

2. If a node v did not receive any message until round $4 \log n' - \lceil r_v \rceil$, it creates the message $(r_v, v, 0)$ that contains its random value r_v, its identifier, and a hop counter. Finally, node v delivers this message to itself together with all other messages received in this round.

3. Once a node v receives one or more messages of the form (r_u, u, x_u) with $u \in V$, it joins a cluster. Let u^* be node that maximizes the term $r_{u^*} - x_{u^*}$ among all received values. Then, v joins the cluster of u^*. If $u^* = v$, it declares itself the cluster head. Otherwise, v joins u^*'s cluster. For this, it stores the identifier of u^* and the identifier of the neighbor $p(v)$ from which it received the message (r_{u^*}, u^*, x_{u^*}). In other words, we declare node $p(v)$ to be its *predecessor*. If v received the message simultaneously from more than one neighbor, it picks the one with the smallest identifier as its predecessor. In the end, v sends $(r_{u^*}, u^*, x_{u^*} + 1)$, i.e., the same message with an increased hop counter, to all its neighbors.

After each node joins a cluster, all nodes add the edge to their predecessor to the spanner (if they have one). By *adding an edge to w to the spanner*, we mean that a node v locally marks w as a neighbor in the spanner and sends a message to w, so w also locally marks v as a neighbor in the spanner. Thus, the edges are always bi-directed and we only add edges that are present in the initial graph G_C. The latter is very important for our construction, as we will require local communication on these edges later on.

However, the subgraph implied by the edges added during this phase is not necessarily connected. These edges only connect a node with its predecessor in its cluster. As we will see, these edges imply a spanning tree rooted in the node's respective cluster head, but there are no paths to nodes in other clusters. Thus, in the second part, we add edges between the clusters to the spanner to ensure it is connected. To simplify notation, we refer to all clusters that contain at least one neighbor of a node as its *neighboring clusters*. We create the edges between clusters in the following way. First, all nodes send the identifier of their cluster head to all their neighbors to inform them about their neighboring clusters. This way, each node can determine all its neighboring clusters and which of its neighbors are in which neighboring clusters. Second, all nodes add an edge to exactly one node of each neighboring cluster. To be precise, suppose that $v \in V$ is in the cluster of some node $u_v^* \in V$ and a neighbor $w \in N_v$ is in the cluster of some node $u_v^* \neq u_w^* \in V$. Then, after receiving the identifier u_w^* from w, node v adds an edge w to the spanner. If more than one neighbor is in this cluster, node v again picks the neighbor with the smallest identifier and only adds the edge to this neighbor to the spanner. As a result, it is also possible that that w perhaps adds another edge to another member of v's cluster. However, the number of edges each individual node adds is bounded by the number of their neighboring clusters. This concludes the construction of the spanner and the first phase of our preprocessing algorithm.

(Step 2) Transform $S(G_C)$ into a bounded degree graph $H(G_C)$: Now, we will construct a bounded degree graph $H(G_C)$ from $S(G_C)$. Note that $H(G_C)$ — in contrast to $S(G_C)$ — is *not* a subgraph of G_C and contains additional edges. Although $S(G_C)$ has few edges in total, there can still be high-degree nodes. Our goal is for high-degree nodes to redirect their edges to other nodes to balance the degrees. This technique is conceptually similar to constructing a child-sibling tree as in [AW07] and [GHSS17].

1. In the first step, we add an *orientation* to the edges similar to the Nash-Williams Forest Decomposition, which is often utilized in algorithms for sparse graphs [AGG+19b, BE10]. This procedure adds an orientation to each edge $\{v, w\}$ such that it is either oriented towards v or w. We will slightly abuse notation and refer to all edges oriented towards a node $v \in V$ as its *incoming* edge and all others as *outgoing* edges. Note that we will still require bidirectional communication between v and w.

   It is well known that any graph of arboricity a has an orientation where each node has at most $O(a)$ outgoing edges [AGG+19b, BE10]. However, instead of directly bounding our spanner's arboricity and using standard techniques to create the orientation, we take a slightly different path. Given the clustering from the previous step, a *sufficiently good* orientation can be constructed in a single round. Every node $v \in V$ declares all edges it added during spanner construction as outgoing edges. Recall that this includes the edge to the predecessor and an edge to one node of each neighboring cluster. The nodes inform their neighbors about the orientation by sending a message containing their identifier.

2. Next, we delegate all incoming edges away and create line-like connections between all incoming nodes. For the construction, consider a node $v \in V$ and let $w_1, \ldots, w_k$ be all nodes with $(w_i, v) \in$

$S(E_C)$, i.e, the incoming edges of v. W.l.o.g., assume that $w_1, \dots, w_k$ are ordered by increasing identifier. Then, for each $i > 1$, v sends the identifier of w_i to w_{i-1} and vice versa.

One can easily verify that each node has at most one incoming edge left (i.e., the edge from w_1 to v) and received at most two edges for *each* outgoing edge (i.e., the edges to w_{i-1} and w_{i-1}).

3.2 Analysis

In this section, we will prove Lemma 3.1. Together with the algorithm from Theorem 1, this implies Theorem 2. In order to prove Lemma 3.1, we need show that it holds:

Lemma 3.2. *The algorithm in the first step creates a connected subgraph $S(G_C)$ of G_C in $O(\log n')$ rounds where each node has at most $O(\log n)$ neighboring clusters w.h.p.*

Equipped with this lemma, we can prove Lemma 3.1 in a straightforward way. Note that each node has at most $O(\log n)$ neighboring clusters, w.h.p., by Lemma 3.2. Therefore, it has at most $O(\log n)$ outgoing edges after computing the orientation in the second step. Thus, the resulting graph $H(G_C)$ has a degree of $O(\log n)$ as claimed. This follows because, in the construction of the child-sibling tree, a node gains two edges, to its predecessor and to its successor, per outgoing edge and loses all but one incoming connection. Further, this part of the algorithm only requires $O(1)$ rounds of communication between neighboring nodes. Together with the $O(\log n')$ rounds from the first step, this proves the theorem.

In the remainder of this section, we will prove Lemma 3.2. We begin with the runtime. It is fairly straightforward to see that both step takes $O(\log n')$ communication rounds. For the clustering phase, note that a node $v \in V$ joins a cluster at the latest in round $4 \log n' - \lceil r_v \rceil$ when it creates its own message. Thus, after $4 \log n' + 1$ steps, all nodes joined a cluster due to our choice of the r_v's. Therefore, the runtime of the first phase is only $O(\log n')$. In the second phase, all nodes only exchange two messages with their neighbors in $S(G_C)$, so its runtime is $O(1)$. Since all nodes know the same estimate of $O(\log n')$, the phases can be synchronized via round counters. Thus, it remains to show that the subgraph created by our procedure is indeed connected and there are *few* edges between the clusters.

In the following, we will show that the resulting graph $S(G_C)$ is connected, and each node has at most $O(\log n)$ neighboring clusters, w.h.p. For the most part, our proof will follow the ideas of Miller et al.'s spanner construction [MPVX15a]. However, we will use some of the insights by Elkin and Neiman [EN18a] and technical details from Haeupler and Li [HL18], who present a similar algorithm explicitly tailored to the CONGEST model. Note that Haupler and Li use the geometric distribution, but this does not make too much of a difference. The key difference — besides the use of the exponential distribution — between our algorithm and the construction by Haeupler and Li is that we broadcast the values for only $O(\log n')$ and not $O(\log n)$ rounds, and redraw r_v's that are larger than $O(\log n')$. Therefore, our algorithm creates slightly different spanners but allows us to reuse some analytical results from Elkin and Neiman. Finally, we want to remark that Elkin and Neiman only described the algorithm differently without the concept of clusters. However, the end result is basically the same as we will see in our analysis.

First, we show that the connectivity follows directly from our construction of the clusters. We begin with the connections within a cluster. Intuitively, the path along the predecessors leads to the head of the cluster. For completeness, we formally show the following claim:

Claim 5. *Suppose that $v \in V$ joined the cluster of $u^* \in V$, then there is a bi-directed path from v to u^* in $S(G_C)$.*

Proof. Recall that a node $w \in V$ joins u^*'s cluster by receiving a message of the form (r_{u^*}, u^*, x_{u^*}). Here, x_{u^*} is the hop counter that is increased each time the message is forwarded. For each node $w \in V$ that is in u^*'s cluster, we can therefore define $x^w_{u^*} \geq 0$ as the value of the hop counter when it joins the cluster[3].

Equipped with this definition, we can now prove the lemma via an induction over all possible values of $x^v_{u^*}$. For the start of the induction, suppose that $x^v_{u^*} = 0$. Note that $x^v_{u^*} = 0$ can only hold if $v = u^*$. Recall that the message (r_{u^*}, u^*, x_{u^*}) that lets nodes join the cluster originates at u^* with a hop counter of 0. The hop counter is increased every round, so u^* is the only node that can ever receive $(r_{u^*}, u^*, 0)$. Thus, for $x^v_{u^*} = 0$, it must hold $v = u^*$ and there is a trivial path contained in the spanner as a node has a path to itself.

Now we get to the induction step. Assume that $x^v_{u^*} = i$ and for all nodes $w \in V$ with $x^w_{u^*} = i - 1$ there is a path to u^* in the spanner. We claim that one of these nodes must be v's predecessor $p(v)$. First, we observe that node $p(v)$ must be part of u^*'s cluster as each node only forwards the message of the cluster it joined. Therefore, the value $x^{p(v)}_{u^*}$ is well-defined. According to the algorithm, $p(v)$ joins the cluster when it receives $(r_{u^*}, u^*, x^{p(v)}_{u^*})$ and then fowards $(r_{u^*}, u^*, x^{p(v)}_{u^*} + 1)$ to v. Upon receiving this message, v joins the cluster. This implies that for v and its predecessor $p(v)$, it holds $x^v_{u^*} = x^{p(v)}_{u^*} + 1$. Therefore, we have $x^{p(v)}_{u^*} = i - 1$ as we assumed $x^v_{u^*} = i$. Finally, v adds a bi-directed edge to $p(v)$ and $p(v)$ has a bi-directed path to u^* per induction hypothesis, so the lemma follows. □

Note that this lemma directly implies that the spanner is connected. Consider an edge $(v, w) \in E_C$. If both nodes are in the same cluster of some node u^*, the lemma certifies that for both nodes, there is a bi-directed path to u^* connecting them. Otherwise, there are two possibilities if the nodes are in different clusters. Either v directly adds the edge (v, w) to the spanner, or it adds another edge (v, w') where $w' \in N_v$ is in the same cluster as w. In the latter case, there must be a path from w to w' by the same argument as before, as they are in the same cluster. Thus, for each edge $(v, w) \in E$, there is a path that connects v and w in $S(G_C)$.

Next, we consider the number of neighboring clusters, which is a bit more involved. Before we go into the details of the analysis, we first introduce some helpful definitions. First, recall that we denote the distance, i.e., the number of hops on the shortest path between two nodes v and w in G_C, as $\mathsf{dist}(v, w)$. As we only consider unweighted graphs, all distances are integer and we have $\mathsf{dist}(v, v) := 0$ and $\mathsf{dist}(v, w) \geq 1$ for any $w \neq v$. Further, for all nodes $u \in V$, we define the value

$$m_u(v) := r_u - \mathsf{dist}(u, v). \tag{3.1}$$

[3] Later on, we will see that it actually holds $x^w_{u^*} = \mathsf{dist}(u^*, w)$, but this is immaterial for this proof.

We denote $m(v)$ to the maximum among these values and $u_v^* \in V$ as the node that drew the respective variable. In other words, it holds:

$$m(v) := m_{u_v^*}(v) := \max\{m_u(v) \mid u \in V\} \tag{3.2}$$

These values will be integral to our analysis as we will see u_v^* is indeed the node that takes v into its cluster. Equipped with these definitions, we show the following.

Lemma 3.3. *Fix a node $v \in V$ and let u_v^* be the node that maximizes $m(v)$. Then, v receives the message $(u_v^*, r_{u_v^*}, \mathsf{dist}(v, u_v^*))$ in round $\left(4\log n' - \lceil r_{u_v^*} \rceil\right) + \mathsf{dist}(v, u_v^*)$ and joins the cluster of u^*.*

Proof. For this proof, we will drop the subscript from u_v^* and simply write u^* if clear from the context. Note that $(4\log n' - \lceil r_{u^*} \rceil) + \mathsf{dist}(v, u^*)$ is the earliest round in which the message can possibly be received by node v as the message is started in round $(4\log n' - \lceil r_{u^*} \rceil)$ and takes (at least) $\mathsf{dist}(v, u^*)$ steps to reach v. First, we show that v cannot receive another message in an earlier round.

Claim 6. *v will not receive any message before round* $(4\log n' - \lceil r_{u^*} \rceil) + \mathsf{dist}(v, u^*)$.

Proof. We begin the proof by noting that u^* minimizes $(4\log n' - r_u) + \mathsf{dist}(v, u)$ among all nodes of V. This can be shown through an elementary calculation. We start with:

$$(4\log n' - r_{u^*}) + \mathsf{dist}(v, u^*) < (4\log n' - r_u) + \mathsf{dist}(v, u) \tag{3.3}$$

To simplify the term, we subtract $4\log n'$. Then, we get:

$$\begin{aligned} &- r_{u^*} + \mathsf{dist}(v, u^*) < -r_u + \mathsf{dist}(v, u) &(3.4)\\ \Leftrightarrow & r_{u^*} - \mathsf{dist}(v, u^*) > r_u - \mathsf{dist}(v, u) &(3.5)\\ \Leftrightarrow & m_{u^*}(v) > m_u(v) &(3.6)\end{aligned}$$

Let $U \subset V$ be the set of nodes whose messages reach v first (note that several such messages may be received in the same round from different neighbors of v). Denote the round in which v first receives something as t. For all nodes $u \in U$, it holds:

$$t = (4\log n' - \lceil r_u \rceil) + \mathsf{dist}(v, u) \tag{3.7}$$

We argue that u^* must be contained in U. For contradiction, suppose that the value would be received in a later round, i.e., it holds:

$$\begin{aligned} t + 1 &\leq (4\log n' - \lceil r_{u^*} \rceil) + \mathsf{dist}(v, u^*) &(3.8)\\ &\leq (4\log n' - r_{u^*}) + \mathsf{dist}(v, u^*). &(3.9)\end{aligned}$$

Now, note that for all values $u \in U$, it holds that $\lceil r_u \rceil - r_u < 1$. Therefore, we have:

$$\begin{aligned}
(4\log n' - r_u) + \mathsf{dist}(v,u) &= (4\log n' - \lceil r_u \rceil + \lceil r_u \rceil - r_u) + \mathsf{dist}(v,u) && (3.10)\\
&= (4\log n' - \lceil r_u \rceil) + \mathsf{dist}(v,u) + (\lceil r_u \rceil - r_u) && (3.11)\\
&< (4\log n' - \lceil r_u \rceil) + \mathsf{dist}(v,u) + 1 && (3.12)\\
&= t + 1 && (3.13)
\end{aligned}$$

By combining these two observations, we see that u^* would not be the node with minimal value as any node in U has a smaller one. This is a contradiction as we defined u^* to be minimal. □

We will see that our lemma holds if all nodes on a shortest path from u^* join the cluster of u^* and forward the message. In particular, we claim the following

Claim 7. *Let $l := \mathsf{dist}(u^*, v)$ and let $P = (w_0 := u^*, \dots, w_l := v)$ be any shortest path from u^* to v. For $0 \leq i \leq l$, it holds that w_i joins the cluster of u^* in round $(4\log n' - \lceil r_{u^*} \rceil) + i$.*

Proof. For contradiction, assume that $w_i \in P$ is the first node that does *not* join the cluster. As all nodes $w_0, \dots, w_{i-1}$ until this point joined the cluster of u^*, it holds that w_{i-1} must have sent the message (r_{u*}, u^*, i) in round $(4\log n' - \lceil r_{u^*} \rceil) + (i-1)$. Therefore, w_i must have joined another cluster in or before this round. As w_i cannot have received a message in an earlier round due to Claim 6, w_i must decide for another cluster in round $(4\log n' - \lceil r_{u^*} \rceil) + i$. More specifically, there is a vertex $z \in V$, such that w_i receives (r_z, z, x_z) and it holds:

$$\begin{aligned}
r_z - x_z &> r_{u^*} - x_{u^*} && (3.14)\\
&= r_{u^*} - i && (3.15)\\
&= r_{u^*} - \mathsf{dist}(u^*, w_i) && (3.16)
\end{aligned}$$

As x_z increases on every hop from z to w_i, it is lower bounded by $\mathsf{dist}(z, w)$. This implies that:

$$r_z - \mathsf{dist}(z, w_i) > r_{u^*} - \mathsf{dist}(u^*, w_i) \qquad (3.17)$$

However, this has implications for the value $m_z(v)$. Using our observations, we see that it holds:

$$\begin{aligned}
m_z(v) &:= r_z - \mathsf{dist}(z, v) && (3.18)\\
&\geq r_z - \mathsf{dist}(z, w_i) + \mathsf{dist}(w_i, v) && (3.19)\\
&> r_{u^*} - \mathsf{dist}(u^*, w_i) + \mathsf{dist}(w_i, v) && (3.20)\\
&= r_{u^*} - \mathsf{dist}(u^*, v) && (3.21)\\
&= m_{u^*}(v) := m(v) && (3.22)
\end{aligned}$$

Here, inequality (3.18) follows from the triangle inequality, inequality (3.20) follows from inequality (3.17), and equality (3.21) holds because w_i is on the shortest path from u^* to v. This is a contradiction as $m(v)$ must be bigger or equal to $m_z(v)$ per definition. Therefore, u^* must the node which minimizes $(r_{u^*} - x_{u^*})$ among all values received by w_i. Thus, w_i would not have joined another cluster and must have forwarded u^*'s message. □

Finally, as $v := w_l$ and $l := \mathsf{dist}(u^*, v)$, the lemma follows. □

Next, we observe neighboring clusters of a node v and show that they are bounded by $O(\log n)$ w.h.p. Suppose that $v \in V$ has a neighbor $w \in V$ that joined another cluster. Then, there must a node $u_v^* \neq u_w^* \in V$ that maximizes $r_{u_w^*} - \mathsf{dist}(u_w^*, w)$. Since v and w are neighbors, the value $r_{u_w^*}$ cannot be much bigger than $r_{u_v^*}$ because otherwise, the corresponding message would reach v much earlier and it would join a different cluster. Following this intuition, we show that the following holds:

Lemma 3.4. . *Let $v, w \in V$ be two neighboring nodes in different clusters. Then it holds $m(v) \leq m(w) + 1$*

Proof. Note that it holds

$$\mathsf{dist}(u_v^*, w) \leq \mathsf{dist}(u_v^*, v) + 1 \tag{3.23}$$

due to the triangle inequality and the fact that v and w are neighbors. This implies that:

$$m(w) > m_{u_v^*}(w) := r_{u_v^*} - \mathsf{dist}(u_z^*, w) \tag{3.24}$$

Using inequality (3.23), we see that:

$$\begin{aligned} m(w) &\geq r_{u_v^*} - (\mathsf{dist}(u_v^*, v) + 1) && (3.25)\\ &\geq m_{u_v^*}(v) - 1 && (3.26)\\ &:= m(v) - 1 && (3.27)\end{aligned}$$

This proves the statement. □

Therefore, the number of neighboring clusters of any node $v \in V$ is upper bounded by the number of values $m_{u'} := r_u - \mathsf{dist}(u, v)$ which are close to $m(v)$. Elkin and Neimann analyzed this random value for the case that the random variables are drawn according to the exponential distribution *without* truncation. Note that their lemma is itself based on an observation by Miller et al.[MPVX15a]. They show the following:

Lemma 3.5 (Lemma 1 in [EN18a]). *Let $d_1 \leq \ldots \leq d_m$ be arbitrary values and let $\delta_1, \ldots, \delta_m$ be independent random variables sampled from the exponential distribution with parameter β. Define the random variables $M = \max_i\{\delta_i - d_i\}$ and $I = \{i : \delta_i - d_i \geq M - 1\}$. Then for any $1 \leq t \leq m$,*

$$\mathbb{P}[|I| \geq t] = (1 - e^{-\beta})^{t-1}.$$

Although we draw the variables differently, their bound will be a very good approximation for our algorithm. Now we can show the following lemma:

Lemma 3.6. *Every node has at most $O(\log n)$ neighboring clusters, w.h.p.*

Proof. For a node $v \in V$ let X_v denote the number of its neighboring clusters. It holds:

$$X_v \leq |\{u \in V \mid r_u - \mathsf{dist}(v,u) \in [m(v)-1, m(v)]\}| \tag{3.28}$$

Further, let $\delta_1, \dots, \delta_{n'}$ be a series of independent, exponentially distributed random variables with parameter β. Define $\hat{m}(v) = \max_{u \in V}\{\delta_i - \mathsf{dist}(u,w)\}$ and consider the variable

$$\hat{X}_v = |\{u \in V \mid \delta_u - \mathsf{dist}(v,u) \in [\hat{m}(v)-1, \hat{m}(v)]\}|. \tag{3.29}$$

Note that $\hat{X}_v$ can be analyzed using Lemma 3.5 by choosing the shortest path distances to v as the d_i's and $\hat{m}(v)$ as M. Finally, let $\mathcal{Z}$ be the event that all variables $\delta_1, \dots, \delta_m$ are smaller than $4\log n'$. Then, it holds:

$$\mathbb{P}[X_v \geq c\log n] = \mathbb{P}[\hat{X}_v \geq c\log n \mid \mathcal{Z}] \tag{3.30}$$

Given these definitions, we want to show that there are constants $c, c' > 0$, such that it holds:

$$\mathbb{P}[\hat{X}_v \geq c\log n \mid \mathcal{Z}] \leq \frac{1}{n^{c'}} \tag{3.31}$$

By the law of total probability, it holds:

$$\begin{aligned}\mathbf{Pr}\left[\hat{X}_v \geq c\log n\right] &= \mathbf{Pr}[\mathcal{Z}] \cdot \mathbb{P}[\hat{X}_v \geq c\log n \mid \mathcal{Z}] + \mathbf{Pr}[\neg\mathcal{Z}] \cdot \mathbb{P}[\hat{X}_v \geq c\log n \mid \neg\mathcal{Z}] &(3.32)\\ &\geq \mathbf{Pr}[\mathcal{Z}] \cdot \mathbb{P}[\hat{X}_v \geq c\log n \mid \mathcal{Z}] &(3.33)\end{aligned}$$

Therefore, by solving for $\mathbb{P}[\hat{X}_v \geq c\log n \mid \mathcal{Z}]$, we get:

$$\mathbb{P}[\hat{X}_v \geq c\log n \mid \mathcal{Z}] \leq \frac{\mathbf{Pr}\left[\hat{X}_v \geq c\log n\right]}{\mathbf{Pr}[\mathcal{Z}]} \tag{3.34}$$

Note that the term in the nominator removed the condition that the variables are smaller than $4\log n'$. Thus, it can be bounded with Lemma 3.5. By choosing $\beta = 1/2$ and $t = c_1 \log n$ with $c_1 := e^{-\frac{1}{2}} \cdot c_2 + 1$, we see that:

$$\begin{aligned}\mathbb{P}[\hat{X}_v \geq c_1\log n] &= (1-e^{-\frac{1}{2}})^{c_1 \log n - 1} &(3.35)\\ &= (1-e^{-\frac{1}{2}})^{\left(e^{-\frac{1}{2}} \cdot c_2 + 1\right)\log n - 1} &(3.36)\\ &\leq e^{-c_2 \log n} = n^{-c_2}. &(3.37)\end{aligned}$$

The last inequality followed from the fact that $(1 - 1/x)^x \leq e^{-1}$ for any $x > 0$. On the other hand, we have the following:

$$\mathbf{Pr}[\mathcal{Z}] := \mathbf{Pr}\left[\bigcap_{i=1}^{n'} \delta_i \leq 4\log n'\right] = 1 - \mathbf{Pr}\left[\bigcup_{i=1}^{n'} \delta_i > 4\log n'\right] \tag{3.38}$$

$$\geq 1 - \sum_{i=1}^{n'} \mathbf{Pr}[\delta_i > 4\log n'] \tag{3.39}$$

The last inequality follows from the union bound. Finally, we use that each δ_i is exponentially distributed with parameter $\beta = 1/2$. It holds:

$$\mathbf{Pr}[\mathcal{Z}] \geq 1 - \sum_{i=1}^{n'} e^{-\frac{1}{2}4\log n'} \qquad \triangleright \textit{Def. of } \delta_i. \tag{3.40}$$

$$\geq 1 - \frac{n'}{n'^2} \geq \frac{1}{2} \qquad \triangleright \textit{As } n' \geq 2 \tag{3.41}$$

Note that we can assume $n' \geq 2$ as a component with one node has no edges. Plugging our two insights together, we get the following:

$$\mathbb{P}[\hat{X}_v \geq c\log n \mid \mathcal{Z}] \leq \frac{\mathbf{Pr}\left[\hat{X}_v \geq c\log n\right]}{\mathbf{Pr}[\mathcal{Z}]} \leq \frac{n^{-c_2}}{1/2} = 2n^{-c_2} \tag{3.42}$$

Thus, by a union bound, every node has at most $c_1 \log n$ neighboring clusters with probability at least $1 - n^{-c_3}$ with $c_3 = c_2 - 2$. This proves the Equation (3.31) for $c_1 \geq c_3 \cdot e^{-\frac{1}{2}} + 3$ and therefore the lemma. □

This proves Lemma 3.1 and therefore Theorem 2!

4
Fast Construction of Spanning Forests

In this section, we will show how the algorithm of Theorem 2 can be used to construct a spanning forest of the (undirected version of the) initial graph G. Given a graph $G = (V, E)$, a spanning tree $S = (V, E_S)$ with $E_T \subset E$ is a subgraph of G with $n - 1$ edges that connects all its nodes. If a graph is not connected, the collection of spanning trees of all its connected components is called a spanning forest. For simplicity, we assume that the input graph is connected; our algorithm can easily be extended to also compute spanning forests of unconnected graphs by running it in each connected component. In particular, we show the following theorem:

Theorem 3. *Spanning Tree in HYBRID*

Let $G = (V, E)$ be a weakly connected directed graph. There is a randomized algorithm that constructs a spanning tree of (the undirected version of) G in $O(\log n)$ rounds, w.h.p., in the HYBRID model.

Each node sends at most $\gamma \in O(\log^3 n)$ bits per round in global mode, w.h.p., and $\lambda \in O(\log n)$ bits along each edge in the local mode.

We describe the algorithm behind this theorem in Section 4.1, and in Section 4.2, we prove Theorem 3.

4.1 Algorithm Description

In this section, we describe the algorithm behind Theorem 3. The description of the algorithm requires some familiarity with the algorithms from Theorems 1 and 2. We refer the reader to Chapters 2 and 3 for the details (and advise to read them first). Having this disclaimer out of the way, we note that the algorithm of Theorem 2 constructs a graph G_L that results from $L \in O(\log n)$ iterations of CreateExpander of the graph G_0 that resulted from Lemma 3.1. This graph has diameter $O(\log n)$ and degree $O(\log^2 n)$, w.h.p. We will use this graph as the starting point for our construction, not the final well-formed tree (i.e., we skip the last step where we reduce the degree).

Given the graph G_L created by CreateExapander, we construct a spanning tree S_L of G_L by performing a BFS from the node with the lowest identifier and then compute an ordering on S_L. Our idea is to iteratively replace all the edges of S_L by edges of G_{L-1}, replace these edges by edges of G_{L-2}, and so on until we reach a graph that contains only edges of G_0. More precisely, our algorithm executes the following steps.

(Step 1) Create a Locally Checkable Ordering: Let v_0 be the node with the lowest identifier. We first perform a BFS in G_L from v_0 and create a BFS-tree S_L. This is our initial spanning tree. Next, we assign a label $l(v)$ to each node $v \in V$. Each node $v \in V \setminus \{v_0\}$ stores the round in which the broadcast reached it, i.e., its distance to v_0, as its label. Note that there can be nodes with equal labels. However, it holds that the root v_0 is the only node with label $l_{v_0} = 0$, all nodes have a parent with a lower label (as the broadcast reached them in the previous round).

(Step 2) Recursively Replace Edges: Next, we want to replace all edges of S_L with edges of G_0 in a recursive fashion. Suppose we are in the i^{th} step of the recursion, i.e., we want to construct $S_{L-(i+1)}$ from S_{L-i}. Let the edge $(v, w) \in S_{L-i}$ be the result of a random walk $(v, v_1, \dots, v_{\ell-1}, w)$ in our CreateExpander algorithm. We assume for the moment that v knows all nodes of the walk and can communicate with them. Then, we perform the following steps:

1. v sends the label $(l(v) \circ j)$ to each v_j on the walk. Here, $\circ$ denotes the concatenation operator for two bit strings.
2. v_j then picks the *minimum* of all its received labels as its new label and informs all its neighbors in $G_{L-(i+1)}$ about it.
3. The root v_0 sets its label to $l(v_0) = l(v_0) \circ 0$ (which therefore remains the unique minimal label). Finally, each node except the root picks an edge (v, w') with $l(w') \leq l(v)$ and adds it to $S_{L-(i+1)}$.

(Step 3) Replace Spanning Tree Edges: Recall that an edge $\{u, w\}$ in S_0 may not exist in G, i.e., if it resulted from a redirection of an edge $\{u, v\}$ in G_0 in the algorithm of Chapter 3, where u and w were incoming nodes of v. So S_0 may not be a spanning tree of G. Now, as before, we can simply model each edge $\{u, w\}$ that does not exist in G as the result of the walk that crossed $\{u, v\}$ and then $\{v, w\}$ in G. This is not a random walk, but that is immaterial. Node u computes the new labels exactly as before and sends them to v and w. Given the labels, we can construct the tree as before.

4.2 Analysis

In this section, we will show that our algorithm creates a spanning tree of G with the time and communication bounds from Theorem 3. We begin with the algorithm's correctness and show that it does indeed construct a spanning tree of G. Recall that the algorithm creates the spanning trees based on a labeling that updates every recursion. In the following, if a node $v \in V$ has edge $\{v, w\} \in E$ to some node w with $l(w) < l(v)$, we call this a critical edge. Note that our algorithm creates the spanning trees $S_L, S_{L-1}, \dots$ by letting each node except v_0 pick a critical edge. Our goal is to exploit the following simple fact:

Lemma 4.1 (Ordering implies Spanning Tree). *Let $G := (V, E)$ be an undirected graph and $l(v_0), \dots, l(v_n)$ be an ordered labeling of the nodes. Further, let E_S be a set of edges. Suppose it holds that*

1. *There is exactly one node v' with $l(v') < l(v)$ for all $v \in V \setminus \{v'\}$, and*
2. *all other nodes $v \in V \setminus \{v'\}$ have a critical edge.*

Then, if each $v \in V \setminus \{v'\}$ adds a critical edge to E_S, the resulting graph (V, E_S) is a spanning tree of G.

Proof. Since each node (except v') adds at most one edge, the resulting graph can have at most $n - 1$ edges and is either a tree or a forest. Therefore, it remains to show that it is connected and, therefore, a tree. However, this follows because each node must have a path to v'. Otherwise, there either would be a node v with $l(v) > l(v')$ that did not add an edge, or there are two nodes with minimal labels. Both options contradict one of the two conditions; thus, our claim follows. ☐

We will show that our algorithm maintains a labeling with the required properties. In other words, in each step of the recursion, each node maintains a label $l(v)$ such that each node v with $l(v) \neq l(v_0)$ has at least one critical edge and there is a unique node v_0 with minimal label. Since all nodes initially use their hop distance to the BFS root v_0 as their label, this condition is trivially fulfilled for S_L. So, we only need to prove that our update rule maintains it. To this, we prove:

Lemma 4.2. *Suppose that S_{L-i} is spanning tree of G_{L-i}, then $S_{L-(i+1)}$ is spanning tree of $G_{L-(i+1)}$.*

Proof. Since there is clearly only one root with a unique minimal label by construction, we must only show that all other nodes have a critical edge. In particular, if some $v \in V$ is not the root, there must be a neighbor in $G_{L-(i+1)}$ with a smaller label. Let now $l'_v := l(x) \circ j$ the minimal label that was assigned to v, i.e., v is the j^{th} node on some walk $(x := v_1, \dots, v_j = v, \dots, v_\ell := y)$ from x to y in $G_{L-(i+1)}$. Consider the following two cases:

- If $x = v$ for some walk $(x, \dots, y)$, then v assigned a label to itself, so the prefix of the new label is its old label $l(v)$. However, since $v \neq v_0$, it must have a parent $p(v)$ in S_{L-i}. Therefore, it must have received a label $l'_{p(v)}$ with prefix $l(p(v))$ from $p(v)$. Since $(v, p(v))$ was a critical edge by construction, it holds $l(p(v)) < l(v)$, i.e., the label $l'(p(v))$ sent by $p(v)$ is strictly smaller than $l'(v)$. Thus, if $v = x$ for some walk, v did not pick the minimum label, which is a contradiction.

- Otherwise, if $x \neq v$, there must be well-defined predecessor v_{j-1} in the walk that is a neighbor of v in $G_{L-(i+1)}$. This predecessor can only get a label smaller or equal to $l(x) \circ j - 1$ which therefore is smaller than v's new label $l'(v)$.

Thus, each $v \in V$ has a neighbor that has a critical edge or is the root (which has the unique minimal label). By Lemma 4.1, we can construct $S_{L-(i+1)}$ in one step by picking a critical edge. □

Thus, the algorithm creates a spanning tree for G as desired. This proves the algorithm's correctness. Therefore, to prove Theorem 3, it only remains to argue that the algorithm has the claimed runtime and communication bounds. We do so in the following lemma.

Lemma 4.3. *The algorithm can be implemented in $O(\log n)$ local communication and $O(\log^3 n)$ global communication, w.h.p.*

Proof. As mentioned, our algorithm requires that the two endpoints of every edge e need to know the nodes that the corresponding walk traversed (i.e., the nodes that *make up* e) while sending no more than $O(\log^2 n)$ messages of size $O(\log n)$ in global mode and $O(1)$ messages per edge in the local mode.

To implement this behaviour, we slightly adapt CreateExpander. As a first change, we add the identifier of each traversed node to each token. Further, when creating an edge, *all* of these identifiers are sent back to the node that started the token ,i.e., we sample the whole walk instead of just the endpoint.

Note that as the token traverses $\ell \in O(1)$ nodes, the size of each token stays (asymptotically) the same. Thus, the endpoints of each edge e of S_{L-i} can inform the endpoints of all edges of $G_{L-(i+1)}$ that make up e. Recall that a node needs to send at most ℓ labels for each edge, resulting in at most $O(\ell \log^2 n)$ labels. However, the labels now not only contain the node's identifier but also the additional information we add in each step. To be precise, the size of a label grows by an additive $\log \ell \in O(1)$ bits for each recursion as we always add a constant number between 0 and ℓ to an existing label. So, after $L \in O(\log n)$ recursions, the label size is still

$$O(\log n) + L \cdot \log \ell = O(\log n). \tag{4.1}$$

Thus, each message message contains $\ell \in O(1)$ labels of size $O(\log n)$ bits. Therefore, a global capacity of $O(\log^2 n)$ messages (or equivantly $O(\log^3 n)$ bits) still suffices.

In the last step, we use the local mode to assign the labels. Here, each node v can be part of $O(d(v))$ walks of length three. These walks only contain neighbors of the node in the original graph G. Since we can now use the local CONGEST edges, a node v can send and receive a new label to and from all its neighbors in one round. Thus, this step can the implemented with $O(1)$ messages or $O(\log n)$ bits of local communication per edge.

Finally, since each recursion step takes 2 rounds of exchanging messages and labels, we finish after $L \in O(\log n)$ rounds. □

Together, these lemmas prove Theorem 3!

5

An $O(\log \Delta + \log \log n)$-Time Algorithm for MIS

In this section, we present a Maximal Independent Set (MIS) algorithm in the HYBRID model with polylogarithmic local and global communication complexity. To the best of our knowledge, this is the first and only such algorithm in the HYBRID model with these restrictions. The MIS problem is defined as follows:

Definition 5.1 (Maximal Independent Set (MIS)). *Let $G := (V, E)$ be an undirected graph, then $S \subseteq V$ is an MIS if and only if it fulfills the two following properties:*

1. *No two nodes in S are adjacent in the initial graph G.*
2. *Every node $v \in V \setminus S$ has a neighbor in S.*

Due to its simplicity, MIS is often regarded as a *benchmark* problem for (new) models of computation. It essentially clarifies how fast one can *break the symmetry* in a graph. That being said, we first take a look at existing bounds for the problem. By a result of Kuhn et al.[KMW04], there are graphs of degree Δ in which computing the MIS takes $\Omega\left(\frac{\log \Delta}{\log \log \Delta}\right)$ rounds, even in the LOCAL model. This was later improved to $\Omega(\min\{\Delta, \frac{\log \log n}{\log \log \log n}\}$ by Balliu et al. [BBH+21]. In terms of upper bounds, there has been a lot of progress in recent years [GGR21, RG20, GG23, GG24] The current upper bound in LOCAL model is of approximately $O(\log \Delta + \log^{5/3} \log n \cdot \log^c \log n)$ [GG24]. In models in which the communication is not limited to the neighbors of a node (which roughly corresponds to our notion of *global communication*), the runtime is often *exponentially* better. For example, both in the CONGESTED CLIQUE and the MPC model [GGJ20, BBD+19, GGK+18, BFU19] one can achieve runtimes of $O(\log \log n)$ or even $O(\log \log \Delta)$ which beat these lower bounds. However, these models allow for communication primitives beyond our model's capabilities. In the CONGESTED CLIQUE, a node can send a message of size $O(\log n)$ to every node in the graph, and in the MPC model, each node can receive $O(n^\delta)$ bits where $\delta < 1$ is a constant. Thus, these extreme improvements are still out of reach. Instead, with limited global communication, we can offer a slight improvement. More precisely, we prove the following theorem:

Theorem 4. *MIS in HYBRID*
Let $G = (V, E)$ be a graph of degree Δ. There is a randomized algorithm in the HYBRID model that computes an MIS of G in $O(\log \Delta + \log \log n)$ rounds, w.h.p. The algorithm requires local capacity $O(\log n)$ and global capacity $O(\log^3 n)$, w.h.p.

Therefore, we only show a marginal improvement compared to state-of-the-are for the LOCAL model. Nevertheless, we can learn something from this. It emphasizes that even a small amount of non-local communication is as strong as unbounded local communication. Further, the algorithm is (arguably) simpler than the state-of-the-art techniques for LOCAL.

Our algorithm combines the so-called *shattering technique* [BEPS16, Gha16] originally developed for the CONGEST model with our overlay construction algorithm. As the name suggests, the shattering technique *shatters* the graph into small components of $O(\Delta \log n)$ undecided nodes in $O(\log \Delta)$ time, w.h.p. Recall that

$$\log(\Delta \log n) = \log \Delta + \log \log n.$$

We then compute a well-formed tree of depth $O(\log \Delta + \log \log n)$ in all these small components. This takes $O(\log \Delta + \log \log n)$ rounds, w.h.p., using the algorithm of Theorem 2. Note that here it is important that the runtime is the logarithmic in the size of the largest component and not n. In each component, we can now independently compute MIS solutions using $O(\log n)$ parallel executions of the CONGEST algorithm by Metivier et al. [MRNZ09]. As we will see, at least one of these executions must finish after $O(\log \Delta + \log \log n)$ rounds, w.h.p, so the nodes only need to agree on a solution for one of the finished executions. The problem is that the nodes cannot *locally* check which executions have finished. Here, the well-formed tree comes into play. We can use it to gather the information of which executions have finished in a single node and then to broadcast information to all nodes in the component in $O(\log \Delta + \log \log n)$ time. This leads to an $O(\log \Delta + \log \log n)$ time algorithm, where Δ is the initial graph's degree.

This chapter is structured as follows: In Section 5.1 we introduce the results by Ghaffari and Metevier et al., which build the basis for the algorithm behind . Given these preliminaries, we present said algorithm in Section 5.2 and analyse it in Section 5.3.

5.1 Preliminaries

Before we discuss our algorithm in detail, we briefly recap two techniques/algorithms used in MIS algorithms. First, we introduce the so-called *shattering technique* and then the bit-optimal CONGEST algorithm by Metevier et al. Both will be important building blocks in our construction.

5.1.1 The Shattering Technique of Ghaffari

Many state-of-the-art MIS algorithms employ the so-called *shattering technique* [BEPS16, Gha16], which conceptually works in two stages[1]: First, there is the so-called shattering stage, where the problem is solved for most nodes using a local strategy. As a result of this stage, each node knows — with probability $1 - o(1/\Delta)$ — whether it is in the MIS itself or has a neighbor in the MIS and, therefore, cannot be in the MIS. We say that these nodes have *decided* as the state of these nodes will not change for the remainder of the algorithm. Likewise, we refer to all other nodes as *undecided* nodes. In the second stage, we solve the problem for all remaining undecided nodes. These undecided nodes only need to communicate with their undecided neighbors as all other nodes know whether they are in the MIS or not. Note that the probability of $1 - o(1/\Delta)$ implies that each undecided node has less than one undecided neighbor in expectation. By a Galton-Watson-like argument, the graph G is, therefore, *shattered* into small isolated subgraphs $G_1, \dots, G_k$ of undecided nodes after the first stage. Two undecided nodes are part of the same subgraph G_i if there is a path of undecided nodes between them. In the second stage, the MIS is solved on these subgraphs where we can exploit that the subgraphs $G_1, \dots, G_k$ are *far* smaller than G.

The first phase can be implemented in (nearly) optimal $O(\log \Delta)$ time due to a brilliant result by Ghaffari[Gha16, Gha19]. It holds:

Lemma 5.1 (Based on Lemmas 4.2 in [Gha16] and 2.1 in [Gha19]). *Let c be a large enough constant and $G := (V, E)$ be a simple undirected graph. There is a distributed MIS algorithm $\mathcal{A}$ with following properties:*

1. *Let B be the set of nodes remaining undecided after $O(c \log \Delta)$ rounds of $\mathcal{A}$. Then, with probability at least $1 - n^{-c}$, all connected components of $G[B]$, the subgraph of G induced by nodes in B, have each at most $O(\Delta^4 \log_\Delta(n))$ nodes.*
2. *Each message contains only 1 bit.*

Given this result, the crux of many modern MIS algorithms lies in their implementation of the second phase. In the LOCAL and CONGEST model, we can use deterministic algorithms to obtain sublogarithmic runtimes. Note that we need to be very careful when we execute randomized algorithms in this phase if the probability of failure only depends on the number of nodes. Since the number of nodes n' in each component is much smaller than n, a bound of $1 - o(n'^{-c})$ does not imply that the algorithm succeeds w.h.p. In models with massive global communication, all remaining nodes and edges of a component can be gathered at a single node using the global communication and then solved locally. This, of course, requires this node to receive a huge amount of messages in a single round. Because of this high message load, this approach cannot be used directly in our model.

5.1.2 The Algorithm of Metevier et al.

Next, we consider MIS algorithms in the CONGEST model. Here, the MIS problem can be solved in $O(\log n)$ time — in expectation and w.h.p. — due to a celebrated algorithm by Luby [Lub86] and Alon et al. [ABI86].

[1]Note that the faster algorithms are more intricate and use more preprocessing stages to reduce degrees, but still rely on this scheme in the end.

The idea behind the algorithms is quite simple: Each node picks a random rank in $[0, 1]$, which is sent to all neighbors. Then, all local minima join the MIS and inform their neighbors about it. All remaining nodes, i.e., nodes that did not join the set and had no neighbor that joined the set, repeat this process until every node has decided. Later, Métivier et al.[MRNZ09] provided a simpler analysis and showed that sending a single bit per round and edge is sufficient. For our algorithm, we take a closer look at the fact that Métivier et al.'s algorithm has an expected runtime of $O(\log n)$. In particular, it holds that in every round, in expectation, half of all edges disappear due to nodes deciding (see [MRNZ09] or the appendix of [Gha16] for a comprehensive proof). Thus, if we execute it on a subgraph with n'^2 edges, it finishes after $O(\log n')$ rounds in expectation. That means, by Markov's inequality, with at least constant probability, the algorithm only takes $O(\log n')$ rounds. In fact, Métivier et al. even prove the following more precise statement:

Lemma 5.2 (Theorem 3 in [MRNZ09]). *There is a randomized distributed algorithm for arbitrary simple graphs $G := (V, E)$ with n nodes in the CONGEST model that:*

1. *finishes in $O(\log n)$ time with probability $1 - o(n^{-1})$, and*
2. *each message contains only 1 bit.*

The lemma implies a success probability of $1 - 1/n'$ if we run the algorithm on a graph of n' nodes. Therefore, if we execute it $O(\log n)$ times independently in parallel, there must be at least one execution that finishes within $O(\log n')$ rounds, w.h.p.

5.2 Algorithm Description

Now we go back to the shattering technique and consider the undecided nodes after the shattering stage. Instead of reporting all edges to an observer that solves the problem locally for each subgraph of undecided nodes, we execute Metevier et al.'s algorithm $O(\log n)$ times in parallel and report which executions finished. Once there is one execution in which all nodes are finished, we signal the nodes to stop via broadcast and let them agree on the outcome of one execution. To do so efficiently, we execute the algorithm of Theorem 2 on each component of undecided nodes. Note that this requires far fewer messages per node than aggregating all edges at a node and still achieves a sublogarithmic runtime.

More precisely, our algorithm to solve the MIS problem operates in the following three steps of length $O(\log \Delta + \log \log n)$ each. To synchronize these steps, we need to assume that, in addition to an approximation of $\log \log n$, the nodes also know an approximation of $\log \Delta$.

(Step 1) Shatter the Graph into Small Components: First, we run Ghaffari's shattering algorithm from [Gha16] for $O(\log \Delta)$ rounds. After executing it, the graph is *shattered* into isolated, undecided components $G_1, \ldots, G_k$. Obviously, the nodes can use the local edges to determine which neighbors are in the same component. The remainder of our algorithm will run on each of these G_i's in parallel.

(Step 2) Construct an Overlay for each Component: Next, we establish a well-formed tree S_i on each G_i using the algorithm of Theorem 2. We denote the resulting trees as $S_1, \dots, S_k$.

(Step 3) Execute Métivier et al.'s Algorithm in Parallel: Using the well-formed tree from the previous step, we can (deterministically) compute aggregate functions on each G_i. Given this powerful tool, we construct an MIS for each G_i as follows:

1. On each G_i, we run the MIS algorithm of Métivier et al. independently $c_1 \log n$ times in parallel for $\tau_{c_2} := c_2 (\log \Delta + \log\log n)$ rounds. Here, c_1 and c_2 are some tunable constants, but c_2 is chosen such that τ_{c_2} is bigger than the diameter of S_i. By Theorem 2, such a minimal c_2 can always be found. Since each execution needs messages of size 1, all messages sent by all executions can be sent in one round of the CONGEST model. More precisely, in each round $r \in [0, \tau_{c_2}]$, a node simply sends the random bit string $M^r(w) := \left(m^r_1(w), \dots, m^r_{c_1 \log n}(w)\right)$ of length $c_1 \log n$ to its neighbor w. Here, the value $m^r_j(w) \in \{0,1\}$ is the 1-bit message that is sent by execution j to w in round r.

2. After τ_{c_2} rounds, each node checks in which executions it has decided, i.e., itself or one of its neighbors joined the MIS. It creates the bit string $F_v^{(0)} := (f_1, \dots, f_{c_1 \log n})$, where $f_j = 1$ if and only if the node v has decided in execution j and $f_j = 0$ otherwise. Again, since there are at most $O(\log n)$ executions, the information on which executions have finished can be fitted into $O(\log n)$ bits and therefore be sent as a single message.

3. We then use S_i to aggregate all executions where all nodes have decided. To be precise, we need to compute the logical AND of all F_v for all $v \in G_i$. Recall that this operation returns 1 if all its inputs are 1 and 0 otherwise. The algorithm to do this is trivial: For τ_{c_2} rounds, each node sends its current value of F_v to all its neighbors and computes the logical AND of all values it received (including its own). More precisely, in round $r > 0$ the current value $F_v^{(r)}$ is computed as follows:

$$F_v^{(r)} := \bigotimes_{w \in N_{S_i}(v) \cup \{v\}} F_w^{(r-1)} \tag{5.1}$$

   Here, the set $N_{S_i}(v)$ contains all of v's neighbors in S_i and $\otimes$ denotes the logical AND. One can easily verify that for a big enough c_2, all nodes know all finished execution after τ_{c_2} rounds. Note that the bits of a finished execution will never be flipped to 0 as all of them are 1. Therefore, we must show that all bits of all non-finished executions are 0, i.e., the value f_j for an execution j will be set to 0 in every node if there is at least one node, say v, that did not decide. A simple induction yields that after r rounds, each node in the distance r to v sets its f_i-value to 0. Since the diameter of S_i is smaller than τ_{c_2}, all nodes know if at least one node in a given execution did not decide.

4. Finally, the nodes adopt the result of an execution that has finished (if there is one). If several executions are finished, the execution with the smallest index is chosen.

5.3 Analysis

In this section, we prove Theorem 4. The runtime and message complexity of the first two steps follow directly from Lemmas 5.1 and Theorem 2.

Lemma 5.3. *The first two phases finish after* $O(\log\Delta + \log\log n)$ *time, w.h.p. and require* $O(\log n)$ *local communication and* $O(\log^2 n)$ *global communication. Further, all resulting trees* $S_1, \ldots, S_k$ *have diameter* $O(\log\Delta + \log\log n)$*, w.h.p.*

Proof. First, recall that we execute the shattering algorithm for $O(\log\Delta)$ rounds. Note this algorithm can be implemented $O(\log\Delta)$ rounds the CONGEST model as it only sends 1-bit messages to each node's neighbors. Thus, the first step requires $O(\log\Delta)$ and $O(1)$ bits of local communication per edge. After these $O(\log\Delta)$ rounds, by Lemma 5.1, the graph is *shattered* into components $G_1, \ldots, G_k$ of size at most $O(\Delta^4 \cdot \log_\Delta n))$, w.h.p. Then, we construct a well-formed tree for each G_i by using the algorithm from Theorem 2. Since each component has size $O(\Delta^4 \cdot \log_\Delta n)$, the construction takes $O(\log(\Delta^4 \cdot \log_\Delta n)) = O(\log\Delta + \log\log n)$ time, w.h.p. Following the theorem, the local message complexity is $O(\log n)$, and the global complexity is $O(\log^3 n)$. Finally, due to the size of each G_i, the resulting trees also have a height of only $O(\log\Delta + \log\log n)$ as claimed. Altogether, this proves the lemma. □

Given the diameter of these trees, we can (deterministically) compute aggregate functions on each G_i in $O(\log\Delta + \log\log n)$ time. To prove Theorem 4, we therefore need to show that in all components, there is at least one execution that finishes in $O\left(\log\Delta + \log\log n\right)$ time, w.h.p. More precisely, we show:

Lemma 5.4. *In each component, there is at least one execution of Métivier et al.'s algorithm that finishes after* $\tau_{c_2} := c_2\left(\log\Delta + \log\log n\right)$ *time, w.h.p.*

Proof. For a given component G_i and execution j, let $X_j \in \{0,1\}$ be the random variable that all nodes have decided after τ_{c_2} rounds. Note that each undecided component must have at least 2 nodes: Seeking contradiction, let there be a component consisting of a single node v. If all of v's neighbors have decided and no neighbor joined the MIS, then v can join the MIS. Otherwise, if there is one neighbor in the MIS, then v has decided per definition as it cannot join the MIS. Combining this fact with Lemma 5.2, we have that $\mathbf{Pr}[X_i = 0] \leq \frac{1}{|V_i|} \leq \frac{1}{2}$. Since all executions are independent, the probability that none of the $c_1 \log n$ executions finishes after τ_{c_2} rounds is:

$$\mathbf{Pr}\left[\bigcap_{j=1}^{c_1 \log n} X_j = 0\right] = \prod_{j=1}^{c_1 \log n} \mathbf{Pr}[X_j = 0] \leq \left(\frac{1}{2}\right)^{c_1 \log n} = e^{-\frac{c_1 \log n}{\log 2}} \leq n^{-c_1'} \tag{5.2}$$

Now recall that there are at most n undecided components. Therefore — by the union bound — all components have at least one finished execution. □

Thus, all nodes have decided at the end of Step 3, w.h.p. Note that aggregating the index of the finished execution can be done in $O(\log\Delta + \log\log n)$ time in each S_i. This proves Theorem 4

6 Conclusion to Part I

In this part of the thesis, we answered the following longstanding open question: *Can an overlay network of polylogarithmic degree be transformed into a graph of diameter $O(\log n)$ in time $O(\log n)$ with polylogarithmic communication?* Additionaly, we derrived result for basic connectivity and symmetry-breaking problems.

6.1 Related Work

Let us first take a look at the history of fast overlay construction. To the best of our knowledge, the first overlay construction algorithm with polylogarithmic time and communication complexity that can handle (almost) arbitrary initial states has been proposed by Angluin et al. [AAC+05]. The authors assume a weakly connected graph of initial degree d. If in each round, each node can send and receive at most d messages, and new edges can be established by sending node identifiers, their algorithm transforms the graph into a binary search tree of depth $O(\log n)$ in $O(d + \log^2 n)$ time, w.h.p. A low-depth tree can easily be transformed into many other topologies, and fundamental problems such as sorting or routing can be easily solved from such a structure.

This idea has sparked a line of research investigating how quickly such overlays can be constructed. Table 6.1 provides an overview of the works that can be compared with our result. For example, [AW07] gives an $O(\log n)$ time algorithm for initial graphs with outdegree 1. If the initial degree is d and nodes can send and receive $O(d \log n)$ messages, there is a deterministic $O(\log^2 n)$ time algorithm [GHSS17].

Result	Runtime	Init. Topology	Communication	Comment
[AAC+05]	$O(\Delta + \log^2 n)$ w.h.p	Any	$O(\log n)$	First Result
[AW07]	$O(\log n)$ w.h.p	Outdegree 1	$O(\log n)$	Opt. for Topology
[JRS+14]	$O(\log^2 n)$ w.h.p	Any	$O(n)$	Self-stabilizing
[GHSS17]	$O(\log^2 n)$	Any	$O(\Delta \log n)$	Deterministic
[GHS19b]	$O(\log^{3/2} n)$ w.h.p	Any	$O(\Delta \log n)$	Previous Best
[GPRT20]	$O(\log^2 n)$ w.h.p	Any	$O(\Delta \log n)$	Churn-resistant
[ACC+20]	$O(\log n)$	Line Graph	$O(\log n)$	Opt. for Topology
Theorem 1	$O(\log n)$ w.h.p	Any	$O(\Delta \log n)$	Chapter 2
Theorem 2	$O(\log n)$ w.h.p	Any	$O(\Delta + \log^2 n)$	Chapter 3

Table 6.1: An overview of the related work. Note that Δ denotes the initial graph's degree. Communication refers to the number of messages of size $O(\log n)$ per node and round.

The research on overlay construction is not limited to these examples. Since practical overlay networks are often characterized by dynamic changes coming from *churn* or *adversarial behavior*, many papers aim to reach and maintain a valid topology of the network in the presence of faults. These works can be roughly categorized into two areas. On the one hand, there are so-called *self-stabilizing* overlay networks, which try to detect invalid configurations locally and recover the system into a stable state (see, e.g., [FSS20] for a comprehensive survey). However, since most of these solutions focus on a very general context (such as asynchronous message passing and arbitrary corrupted memory), only a few algorithms *provably* achieve polylogarithmic runtimes [JRS+14, BGP13], and most have no bounds on the communication complexity. On the other hand, there are overlay construction algorithms that use only polylogarithmic communication per node and proceed in synchronous rounds. Gilbert et al.[GPRT20] combine the fast overlay construction with adversarial churn. They present a construction algorithm that tolerates adversarial churn as long as the network remains connected, and there eventually is a period of length $\Omega(\log n^2)$ where no churn happens. The exact length of this period depends on the goal topology. Further, there is a paper by Augustine et al. [ACC+20] that considers $\widetilde{O}(d)$-time[1] algorithms for so-called graph realization problems. They aim to construct graphs of *any* given degree distributions as fast as possible. They assume, however, that the network starts as a *line*, which makes the construction of the graphs considered in this work considerably more straightforward.

6.2 Applications & Implications

Our results directly come with various applications and implications for several distributed computation problems. We point out the following immediate implications that simply follow from the fact that any initial overlay topology can be turned into a well-formed overlay in $O(\log n)$ time.

Monitoring in Overlays The paper [GHSS17] by Gmyr et al. presents so-called monitoring problems where the goal is to observe the properties of the input graph. *Every* monitoring problem presented in [GHSS17] can now be solved in $O(\log n)$ time, w.h.p., instead of $O(\log^2 n)$ deterministically. These problems include monitoring the graph's node and edge count, its bipartiteness, and the approximate (and even exact) weight of an MST.

[1] $\widetilde{O}(\cdot)$ hides polylogarithmic factors

Bootstrapping Churn-resistant Overlays There is a variety of algorithms maintaining an overlay topology under randomized or adversarial errors. These works focus on quickly reconfiguring the network to distribute the load evenly (under churn) or to reach an unpredictable topology (in the presence of an adversary) [DGS16, AS18, APR+15, GVS19b]. A common assumption is that the overlay starts in some well-defined initial state and the algorithm is granted $O(\log n)$ rounds of *bootstrapping*. These rounds have no churn, so the maintenance protocol can be *prepare* itself. With our algorithm, the former assumption can be dropped as the $O(\log n)$ rounds of bootstrapping suffice to build a suitable overlay from any initial configuration.

Executing NCC Algorithms in the NCC_0 For most algorithms presented for the NCC (and hybrid networks that model the global network by the NCC) [AGG+19b, AHK+20b, FHS20], the rather strong assumption that all node identifiers are known may be dropped. Instead, suppose the initial knowledge graph has degree $O(\log n)$. In that case, we can construct a butterfly network in $O(\log n)$ time, which suffices for most primitives to work (note that all presented algorithms have a runtime of $\Omega(\log n)$ anyway).

Advanced Connectivity Problems Our algorithm for spanning forests can be used to further problems related to connectivity. In [GHSW23], they are used in an algorithm that checks whether a graph is biconnected in time $O(\log n)$. Further, in Schweichart's thesis [Sch23b], they are utilized in an algorithm that checks whether a graph is (approximately) k-connected in $O(\log^3 n)$ time.

6.3 Open Questions & Possible Future Work

We conclude this chapter with possible directions for future work and open problems. Note that any idea sketched in the remainder should not be understood as propositions or even conjectures but rather as educated guesses on which directions could be worthwhile.

Improving the Communication Complexity First, we note while our solution is asymptotically time-optimal, our communication bounds may likely be improved. If the initial degree is d, then our nodes need to be able to communicate $\Theta(d \log n)$ many messages. However, as is implicitly proposed in [AAC+05], an algorithm might only require a communication capacity of $\Theta(d)$. Eradicating the additional $\log n$ factor from our algorithm seems to be non-trivial and poses an interesting goal. Note that our approach crucially requires each node to maintain $\Theta(\log n)$ connections to maintain the overlay's connectivity as we essentially construct a random graph in each iteration. It is well-known that even if all edges are picked uniformly at random, one requires $O(\log n)$ per node to obtain a connected graph, w.h.p. Thus, a time- and communication optimal algorithm must resort to different techniques. We note that it *may* be possible to adapt the PRAM algorithm of Halperin and Zwick [HZ01] to our model, which uses $O(n+m)$ processors to construct a spanning tree in time $O(\log n)$. Similar to [AAC+05, AW07, GHSS17, GHS19b], the idea of the algorithm is to merge supernodes repeatedly. To merge a sufficiently large set of supernodes at once, the authors observe that it suffices to perform $O(\log n)$ random walks of length ℓ to discover $\ell^{1/3}$ many supernodes, w.h.p. If each supernode of size $\Omega(\ell)$, this can be done in $O(\log \ell)$ depth. As in the PRAM model, it is possible to perform such random walks in time $O(\log \ell)$ in overlay networks [DGS16, AS18] under certain conditions. However, this adaption is highly non-trivial and the resulting algorithm will be significantly more complex than our solution.

Archieving Instance Optimal Runtime Recall that our algorithm is optimal if the initial topology is a line consisting of n nodes. However, it is unclear whether it is optimal beyond this worst-case scenario. Notably, Liu et al. [LTZ20] recently proposed an $O(\log \mathrm{HD} + \log \log_{m/n} n)$ time algorithm for computing connected components in the PRAM model. Here, HD is the components' hop diameter. Only for a line, HD happens to be equal to the number of nodes, but generally, it can be much smaller. As this algorithm outputs a spanning tree if the graph has one connected component. Recall that a spanning suffices to construct a well-formed tree through the algorithm of Aspnes and Wu [AW07]. Assadi et al.[ASW19] achieve a comparable result in the MPC model (that uses $O(n^{\delta})$ communication per node) with a runtime logarithmic in the input graph's spectral expansion. Thus, if these algorithms could be transferred to the HYBRID model, they could (possibly) imply faster overlay constructions. Note that, however, the MPC model and the PRAM are arguably more powerful than the overlay network model since nodes can reach *any* other node (or, in the case of the PRAM, processors can contact arbitrary memory cells), which rules out a naive simulation that would have $\Omega(\log n)$ overhead if we aim for a runtime of $O(\log n)$. Also, if the degree is unbounded (our assumption for the HYBRID model), simulating PRAM algorithms, which typically have work $\tilde{\Theta}(m)$, becomes completely infeasible.

Constructing Overlays under Churn Another possibly interesting application of our algorithm is the construction of robust overlay networks under churn, i.e., nodes joining a leaving the network during the construction. One promising approach is ensuring that our algorithm maintains a sufficiently high *vertex expansion* throughout every iteration (instead of conductance). That means each subset of nodes must not only have many edges that lead out of the subset but also must be connected to many *different* nodes to handle a big fraction of nodes leaving. Thus, we must additionally analyze the number of random walk tokens emitting from a given subset that end at the same node. This can likely be done using more advanced spectral and/or combinatorial methods. To illustrate this claim, assume every ℓ rounds a, say, logarithmic fraction of the nodes fail and thereby drop all tokens they received. This, of course, is a very basic churn model, but it is sufficient to convey our point as other churn models are usually stronger and we would face a similar problem. If each node fails independently with probability $p := \frac{1}{\Delta}$, one can easily verify that the *expected* number of outgoing edges in each phase is only decreased by $O(1)$. Thus, in expectation, the algorithm continues to increase the graph's conductance by a constant factor each phase. However, this result does not *trivially* hold with high probability anymore as the individual walks that create the connections do not fail independently of one another. In the extreme case, i.e., if all walks of a single end at the same failing node, the corresponding node does not create a single edge. To mitigate this, we need the know the number of random walks (of a single node or set) that end at distinct nodes. If a constant fraction of walks end at different nodes, then a constant number of the tokens is dropped independently. This is enough for the Chernoff bound to kick in, as we could now show that constant fractions of tokens survive w.h.p. (that means a constant fraction of the independent tokens). However, many details need to be worked out and clarified.

Property Testing Our algorithm has the interesting property that it's runtime is closely tied to the graph's conductance, which makes it much faster on graphs that already have a good conductance. Given this insight, our algorithm can likely be used as a basis for conductance testing the HYBRID model. In conductance testing, for given parameters Φ and ϵ, the goal is to accept that have conductance Φ and reject graphs that are at least ϵ-far from having conductance $O(\Phi^{2+o(1)} \log n)$. Here, the term ϵ-far means that an ϵ-fraction of all edges must be changed to obtain a graph of the desired conductance. In this particular scenario, it suffices to find a subset of size $\Theta(\epsilon n)$ with bad conductance, i.e., $O(\Phi^{2+o(1)} \log n)$, to reject the graph. There are two algorithms [FV18] or [ŁMOS20] that consider this problem in distributed models. For these algorithms, either the runtime or the global communication are in $\widetilde{\Theta}(\Phi)$. Our algorithm — most likely — can reduce this because we can reduce the conductance in a preprocessing step. Following our analysis, any graph of conductance $\Theta(\Phi)$ can be transformed into a constant conductance graph in $O\left(\frac{\log(\Phi)}{\log(\ell)}\right)$ rounds. On the other hand, if we apply our algorithm on a graph with a set with conductance $O(\Phi^{2+o(1)} \log n)$, we observe that in every round, the conductance can only increase by a factor of $O(\ell)$ as on expectation, $O(1)$ random walks enter or leave the subset in expectation via each edge. Thus, after $O(\frac{\log(\Phi)}{\log(\ell)})$ rounds, if we choose ℓ carefully enough, the set likely still has a somewhat bad conductance. Then, one can apply the algorithm of [FV18] or [ŁMOS20] with reduced complexity on the graph created by our algorithm. However, the exact bounds for Φ, and ϵ rely on a very careful choice of ℓ, we defer this application to future work. As before, many details need to be worked out and clarified to obtain a feasible solution.

Minimum Spanning Trees While our algorithm can be used to quickly compute spanning trees, we do not know whether our techniques can also help find minimum spanning trees. There does not seem to be any reason to believe that computing an MST is inherently harder; In fact, it is easily possible with earlier overlay construction algorithms [AGG+19c], but requires $O(\log^3 n \log W)$ time where W is the largest weight. That algorithm follows the approach of merging supernodes, so we cannot trivially *plug in* our algorithms to speed up certain parts. Schweichhart's thesis [Sch23b] has proven that if the maximum weight W is at most $O(\log n)$, our spanning tree can indeed be adapted to construct MSTs faster than [AGG+19c]. However, this bound on W is a severe limitation. It seems that in order to be genuinely faster than [AGG+19c] for all values of W, we would need additional or possibly different techniques altogether. While MST algorithms for more powerful models such as the congested clique or the MPC model [Now19, JN18, GP16] hardly seem applicable, it might be worthwhile to investigate whether PRAM algorithms provide useful techniques for overlay networks [PR02, CHL01].

Part II

Distributed Graph Algorithms for Distance Based Problems

Overview of Part II

In this part of the thesis, we consider distributed algorithms for various graph problems in the CONGEST and HYBRID model. To precise, we consider the following three fundamental problems. We present an algorithm that computes a so-called compact routing scheme for G. These routing scheme allow each node route a message to every other node via a path that is not significantly longer than the shortest path between these two nodes. Further, we present an algorithm for constructing so-called low-diameter decompositions. These are subgraphs of bounded diameter, such that only very few edges have endpoints in different subgraphs. These decomposition are important building blocks in the design of (distributed) divide-and-conquer algorithms. Finally, we present a algorithms for a variant of the dominating set problem where for each node we to mark one node in some distance $\mathcal{D}$ and wish to mark as few nodes as necessary. Again, due to its generality, this algorithm can be used as a building block to other (distributed) optimization problems. We remark that our algorithms work for arbitrary graphs but exhibit much better properties when executed on a restricted graph class.

The results presented here are based on (but significantly extend on) the following publications. First, we presented the basic construction of routing schemes for planar graphs in CONGEST and the HYBRID model the following publication:

> Jinfeng Dou, Thorsten Götte, Henning Hillebrandt, Christian Scheideler, and Julian Werthmann. Brief announcement: Distributed construction of near-optimal compact routing schemes for planar graphs. In Rotem Oshman, Alexandre Nolin, Magnús M. Halldórsson, and Alkida Balliu, editors, *Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing, PODC 2023, Orlando, FL, USA, June 19-23, 2023*, pages 67–70. ACM, 2023

This result mostly combined and generalized known results on planar graphs in the CONGEST and HYBRID model. In the follow-up work, we considered the problem of creating low-diameter decompositions for the larger class of universally k-path separable graphs. It was published in the following paper:

Jinfeng Dou, Thorsten Götte, Henning Hillebrandt, Christian Scheideler, and Julian Werthmann. Distributed and parallel low-diameter decompositions for arbitrary and restricted graphs. In Raghu Meka, editor, *16th Innovations in Theoretical Computer Science Conference (ITCS 2025), New York City, NY, USA, January 7–10, 2025*, Leibniz International Proceedings in Informatics (LIPIcs), page (to appear), Dagstuhl, Germany, 2025. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik

However, the ideas developed for this paper also lead to an improved distributed construction of routing schemes for universally k-path separable and even general graphs. For the latter, the improvement only holds for the HYBRID model. The respective algorithms have not been published before but are basically corollaries of the main results of the two previuously mentioned publications. Finally, the last publication that builds the basis of this part of this part is:

Thorsten Götte, Christina Kolb, Christian Scheideler, and Julian Werthmann. Beep-and-sleep: Message and energy efficient set cover. *Theor. Comput. Sci.*, 950:113756, 2023

The article presents an algorrithm for the Set Cover problem in the so-called BEEPING model. This algorithm builds the foundation for our last result. Note that this is the journal version of the following conference paper

Thorsten Götte, Christina Kolb, Christian Scheideler, and Julian Werthmann. Beep-and-sleep: Message and energy efficient set cover. In Leszek Gasieniec, Ralf Klasing, and Tomasz Radzik, editors, *Algorithms for Sensor Systems - 17th International Symposium on Algorithms and Experiments for Wireless Sensor Networks, ALGOSENSORS 2021, Lisbon, Portugal, September 9-10, 2021, Proceedings*, volume 12961 of *Lecture Notes in Computer Science*, pages 94–110. Springer, 2021

The journal version includes all proofs that were omitted in the conference version and corrects the lower bound presented in the conference version.

The remainder of this part is structured as follows. We begin with some preliminaries in Chapter 7, in which we introduce some auxiliary results. First, we introduce a metamodel, making it easier to describe algorithms that efficiently run in both the CONGEST and the HYBRID model without describing each algorithm twice. Further, we present some simple helper algorithms that we will use as a black box throughout this part. Next, we introduce algorithms for the construction of *so-called* separators that can be efficiently implemented in both CONGEST and HYBRID. Separators are subgraphs that, if removed, decompose the graph into connected components containing a most a constant fraction of the nodes. These separators will be an important building block for our subsequent algorithms. In Chapter 8, we present an algorithm that produces a weaker form of a separator. This separator only ensures that upon its removal, all nodes can reach few nodes in some distance $\mathcal{D}$. As we will see, this is good enough for our application and much easier to compute in distributed setting. In Chapter 9 we present algorithms to construct a proper path separator in planar graphs. In contrast to the previous algorithm, this algorithm computes a proper separator that possibly has further applications.

After these technical chapters introduce general techniques and building blocks, we continue with the three main algorithms. Chapter 10 is dedicated to constructing a low-diameter decomposition for both general and restricted graphs in CONGEST and HYBRID. We show that several sequential clustering algorithms from the literature can (in a weaker form) also be efficiently implemented in CONGEST and HYBRID. Combined with our two separator constructions, this leads to almost optimal algorithms for many graph classes. Chapter 11 presents routing schemes for general and restricted graphs. These schemes also use both the separator construction from Chapters 8 and 9 and the clustering algorithms from Chapter 10. Finally, in Chapter 12, we consider a somewhat orthogonal problem of computing a simple covering. Note that these three problems are relatively disjoint. Therefore, all three chapters have self-contained sections on related work and their own conclusions, including open questions and possible directions for future work. Finally, in Chapter 13

We remark that the papers [DGH$^+$23] and [DGH$^+$25] that focus on routing schemes and decompositions also presented work-efficient algorithms for the PRAM. These results are omitted in this thesis to keep the focus solely on distributed models, i.e., on CONGEST and HYBRID. However, all of our results can also be used to derive PRAM algorithms with near-linaer work and polylogarithmic depth. Finally, there are two important notes regarding the journal article [GKSW23] co-authored with Christina Kolb, Christian Scheideler, and Julian Werthmann: Note that this thesis does not contain all algorithms presented in [GKSW23]. The remaining results can be found in the thesis of Christina Kolb [Kol22]. The thesis also contains a summary of the results we present in Chapter 12 as they form the necessary preliminaries for the results presented in [Kol22]. This summary also includes the proofs, which leads to some textual overlap. However, we emphasize that these passages are only used as *preliminaries* in [Kol22]. Finally, we remark that the author was involved in designing the algorithms, conceiving the analysis, and writing all papers that are the basis for this chapter.

7
Preliminaries for Part II

In this part of the thesis, we consider algorithms for graph problems in the distributed setting, namely the CONGEST and the HYBRID model. Before we get into the details of our algorithms, we use this chapter to present some useful techniques and preliminaries. In particular, we discuss the fundamental impact of a graph's topology on the solution quality and complexity of a distributed algorithm. As such, this chapter mainly presents, aggregates, and unifies known results of graph theory and distributed graph algorithms.

We begin our little tour by reminding ourselves why the graph's topology plays a crucial role in our algorithms and why we do not simply present algorithms that work perfectly well on all topologies. As we have already argued in the introduction, the solution to a graph-based problem depends on the properties of the input graph. For example, many problems that are NP-hard and even hard to approximate in general graphs might be trivial (or at least significantly easier) to solve in trees, planar graphs, graphs of bounded treewidth, etc. See, for example, the survey by Bodleander for an excellent overview on algorithms for graphs of bounded treewidth [Bod93]. When we consider graph-based problems in distributed models like CONGEST and HYBRID where the communication graph is equal to (or at least part of) the input graph, another very important factor comes into play: Some problems cannot be solved *fast* on certain graphs as there exist pathologic communication graphs in which the necessary information to solve a problem cannot be aggregated fast enough. However, other graph classes with favorable topological properties allow specific problems to be solved much more efficiently. To name an extreme example, in a complete graph, many problems can be solved in a sublogarithmic or even constant number of rounds as all nodes directly communicate with one another.

In this chapter, we present useful preliminaries for the algorithms presented in this part of the thesis. First, we present a generic framework that allows us to express complex distributed algorithms through a series of simpler operations, namely aggregations and shortest-path computations. This framework uses known techniques for graph-based distributed problems, namely the so-called minor aggregation framework [GH16b, HWZ21, GZ22a, GH21] and algorithms for $(1+\epsilon)$-approximate shortest paths[REGH22]. For both these operations, near-optimal distributed algorithms are known. In particular, the algorithms are especially fast in restricted graphs. We present the framework and the corresponding algorithms in Section 7.1. Then, in Section 7.2, we present a very helpful lemma by Ghaffari and Zuzic that conveniently enables the design of algorithms on subforests of a given graph.

7.1 A Divide-And-Conquer Theorem for Distributed Algorithms

Divide-and-conquer is a well-known paradigm in algorithm design where the problem is recursively divided into smaller subproblems that are easier to handle. In this section, we present a meta-model for distributed divide-and-conquer algorithms in the HYBRID and CONGEST model that will simplify the description of the forthcoming algorithms. Rather than exploiting these models' intricacies in bespoke algorithms, we reduce our algorithms to $\tilde{O}(1)$ applications of a $(1+\epsilon)$-approximate shortest path algorithm with $\epsilon \in O\left({}^{1}\!/\!_{\log^2 n}\right)$ and some simple aggregations on subsets of the input graph. To be precise, our algorithms are based on approximate set-source shortest paths (SetSSP) and so-called minor aggregations. In the remainder of this section, we will clarify what exactly these operations do and how efficiently they can be implemented in the CONGEST and the HYBRID model.

Following the divide-and-conquer approach, our algorithms will divide the input graph $G = (V, E, w)$ into disjoint connected subgraphs $\mathcal{C}_1, \dots, \mathcal{C}_N$ and solve certain tasks on these subgraphs. While this can be done without repercussions on the runtime in the sequential model, one needs to be careful in distributed models, where an algorithm's runtime is often intertwined with the input graph's topology. To illustrate this, consider the broadcast problem where one node wishes to send a message to all other nodes. Clearly, thus requires at least HD rounds in CONGEST. When there is one node that broadcasts a message to all other nodes in G, it can be implemented by a trivial algorithm: In each round, each node that knows the message forwards it all its neighbors. After HD rounds, each node must know the message; thus, the algorithm is optimal. However, this does *not* trivially imply that a broadcast in a collection of subgraphs $\mathcal{C}_1, \dots, \mathcal{C}_N$ can also be implemented in HD rounds in all subgraphs in parallel. Suppose one node in each subgraph wants to send a single message to all nodes in that subgraph. In this setup, the simple algorithm from above fails. The subgraphs might have a hop-diameter greater than HD, so we cannot confine the algorithm to operate only in their respective subgraphs. Rather, the algorithm must also use nodes and edges from outside the subgraph to achieve a good runtime. But we need to be careful. We cannot let all nodes and edges help with all the broadcast algorithms, as this would create too much congestion. For tasks more complex than broadcast, this becomes even more dire.

For these reasons, we will consider algorithms that consist only of operations that can be efficiently implemented in subgraphs in both CONGEST and HYBRID. In a brilliant series of works [HWZ21, GZ22a, GH21], it was shown that for certain algorithmic tasks, it is indeed possible to solve them in all subgraphs in parallel in $\tilde{O}(\mathrm{HD} + \sqrt{n})$ time in the CONGEST model; in some graphs even in $\tilde{O}(\mathrm{HD})$. In particular, one can compute simple aggregation functions like *sum*, *maximum*, or *average* in the connected subgraphs. That means, each node in the connected subgraph has an input value and all nodes in the subgraph need the learn the result of an aggregation on all these values, i.e., their sum, their maximum, or their average. In HYBRID, the situation is similar. Here, we can use the global communication capabilities to implement simple aggregation in each component $\mathcal{C}_1, \dots, \mathcal{C}_N$ in $O(\log n)$ time, w.h.p. We will refer to these operations as *minor aggregations*. However, these aggregations alone are not quite enough for our algorithms. Further, we will also need to compute the approximate shortest paths in each connected subgraph. To be precise, each connected subgraph $\mathcal{C}_i$ has its own source value s_i, and each node in the subgraph learns its distance to s_i within the subgraph, i.e., only using edges within the subgraph. Here, we can use the state-of-the-art approximate SSSP algorithm from [RGH+22] that computes an approximate shortest path tree rooted in an arbitrary set of nodes. As we will see, it can be implemented in $\tilde{O}(\epsilon^{-2}(\mathrm{HD} + \sqrt{n}))$ time and in some graphs even in $\tilde{O}(\epsilon^{-2} \cdot \mathrm{HD})$ time in CONGEST and $\tilde{O}(\epsilon^{-2})$ time in HYBRID, w.h.p. This algorithm can trivially be extended to run in subgraphs $\mathcal{C}_1, \dots, \mathcal{C}_N$ by giving all edges between subgraphs infinite weight. Thus, just like the aggregations, we can efficiently compute approximate shortest paths in all subgraphs in parallel.

Morally, we will show the following over the course of this section: Let $G := (V, E, w)$ be a (weighted) graph. Further, let $\mathcal{A}$ be an algorithm that can be broken down into τ_s $(1+\epsilon)$-approximate shortest computations with $\epsilon < 1$ and τ_m aggregations. We will show that on K_r-free graphs, if we have $\tau = \tau_s + \tau_m$ steps and compute $(1+\epsilon)$-approximate shortest paths with $\epsilon \in \Omega(1/\log^2 n)$, the overall runtime is $\tilde{O}(\tau\mathrm{HD})$ in CONGEST and $\tilde{O}(\tau)$ in HYBRID, w.h.p. For general graphs, slightly different bounds apply.

Note that this is a simplified version, as the precise technical theorem has some more nuances. In particular, we will clarify the exact definition of aggregations and approximate shortest path throughout this section. Further, we want to emphasize that virtually all the results in this section have been shown in previous works, and we refer to these works for the formal proofs. We merely compile and and combine their results here. This section is structured as follows:

- In Section 7.1.1, we present the Minor Aggregation model, a meta-model that gained a lot of popularity in the study of graph algorithms in CONGEST, but can also be used for efficient algorithm in HYBRID. In this model, complex algorithms are broken down into aggregations on connected subgraphs. These aggregations can then be solved in essentially optimal time in each input graph in CONGEST and in logarithmic time in HYBRID.
- In Section 7.1.2, we gather the state-of-the-art algorithm for $(1+\epsilon)$-approximate shortest paths and discuss its time complexity in various setups.
- In Section 7.1.3, we combine all of our insights into a technical theorem that enables us to conveniently bound the runtime of any divide-and-conquer algorithm that only uses aggregations and shortest path computations in both CONGEST and HYBRID.

7.1.1 The Minor Aggregation Framework

The Minor Aggregation model (MinAgg) is a novel meta-model that allows easy description algorithms to be efficiently implemented in the CONGEST, the PRAM, or even the HYBRID model. Although our algorithms do not exploit this model's full power, we will give a short overview for the sake of completeness. For a more detailed insight into the model and its strengths and caveats, we refer to an excellent series of papers investigating this model in greater depth [HWZ21, GZ22a, GH21].

We begin with a brief history of this very recent model before we proceed with its formal definition to better understand the modelling descisions. Interestingly, the MinAgg model was not originally conceived as a meta-model. Rather, it was used as a convenient way for creating so-called *universally optimal* CONGEST algorithms. To understand the notion of *universally optimality*, we take a short detour and need to dig a bit deeper into the properties and caveats of the CONGEST model. First, we make ourselves clear that it is a fundamental property (one might call it a curse) of the CONGEST model that the communication graph and the problem graph share the same topology. In other words, the graph $G := (V, E, w)$ is not just an input for some algorithmic problem, say, computing shortest paths, but also dictates which nodes may communicate with one another. Thus, the optimal time in which a problem can be solved depends (arguably) much stronger on G's topology than, say, in the PRAM or the HYBRID model, where nodes can communicate independently of the topology. This may lead to some ambiguity when we talk about lower bounds for the runtime of an algorithm. In particular, one has to consider lower bound in runtime in CONGEST with a grain of salt as many of them are simply stated as *worst-case* bounds. In other words, they state that *there is* a certain graph G on which a problem can only be solved in $\Omega(T)$ time. For example, it is well known that the weighted shortest path problem takes at least HD time on any graph G. However, there are pathological graphs where it takes $\Omega(\sqrt{n})$ time, although they have small diameters. Thus, both HD and $\Omega(\sqrt{n})$ may justifiably regarded as lower bounds for the problem.

When designing a distributed algorithm, we obviously want the runtime to be optimal for the specific graph (or at least the graph family) on which we execute it. This brings us to the aforementioned concept of *universally optimality*. A recent line of works that involved the input of many authors took a closer look at this issue and introduced the notion of *universal optimality*. The goal was to design algorithms that are not optimized for the worst-case topology but indeed *always* run time-optimal in their respective communication graph. Their starting point was a paper that dealt with finding the MST of a planar graph in $\widetilde{O}(\text{HD})$ time[GH16a]. This problem was known to require $\widetilde{O}(\text{HD} + \sqrt{n})$ time in general graphs due to the aforementioned pathological graphs. One of the paper's main new insights was the fact that in a planar graph G, one can solve simple problems like computing the minimum or the sum of all values stored by the nodes of (disjoint) connected subgraphs of G in $\widetilde{O}(\text{HD})$ time for all these subgraphs in parallel. Notably, the bound holds even if the diameter of these connected subgraphs is much larger than $\widetilde{O}(\text{HD})$ itself, which makes it highly non-trivial in CONGEST. In other graphs, this is not generally possible and, in fact, can take up to $\widetilde{O}(\text{HD} + \sqrt{n})$ time. Soon, however, it was discovered that not only planar graphs but many other graph classes allow us to solve this problem in time $\tilde{O}(\text{HD})$. To formalize this, they introduced the shortcut quality $\mathcal{Q}(G)$ of a graph G as *the* parameter that captures how complex it is to solve a problem on a specific graph G.

With this background, we will now formally introduce MinAgg framework. Consider a network $G = (V, E)$ and a (possibly adversarial) partition of vertices into disjoint subsets $V_1, V_2, \dots, V_N \subset V$, each of which induces a connected subgraph $G[V_i]$. We will call these subsets *parts*. Further, let each node $v \in V$ have private input x_v of length $\widetilde{O}(1)$, i.e., a value that can be sent along an edge in $\widetilde{O}(1)$ rounds. Finally, we are given an aggregation function $\bigotimes$ like `SUM`, `MIN`, `AVG`, ... An aggregate function $\bigotimes$ maps a multiset $S = (x_1, ..., x_N)$

of input values to some value $\bigotimes(S)$. For some functions $\bigotimes$ it might be hard to compute $\bigotimes(S)$ in a distributed fashion because not all values of S are available at once. Thus, the function must be computed incrementally. Further, the in- and outputs of a function may be too big to be sent a single (or even polylogarithmically many) messages. This is clearly the case for MAX, MIN, and AVG as the result are simply numbers that can be encoded in $O(\log n)$ bits (given that the inputs were logarithmic). However, there are also aggregate functions where this is not the case. For example, a function that gathers all node identifiers in the system is certainly an aggregation function. But the result of this function can only be encoded in $\Omega(n)$ bits. Thus, we may only consider functions where both in- **and** output are of polylogarithmic size. We shall call these *admissible* functions.

Definition 7.1 (Admissible Aggregation Function)**.** *An aggregate function $\bigotimes$ is called admissible if its input and all its intermediary outputs are of polylogarithmic size.*

Having these technicalities out of the way, the goal is to simultaneously compute an admissible aggregation function $\bigotimes_{v \in V_i} x_v$ for each part. Later on, the model was refined to the following definition, we use in this work:

Definition 7.2 (Distributed Minor-Aggregation Model, from [GZ22a])**.** *We are given a connected undirected graph $G = (V, E)$. Both nodes and edges are individual computational units (i.e., have their own processor and private memory). Communication occurs in synchronous rounds. Initially, nodes only know their unique $\tilde{O}(1)$-bit ID and edges know the IDs of their endpoints. Each round consists of the following three steps (in that order).*

- ***Contraction step.*** *Each edge e chooses a value $c_e = \{\bot, \top\}$. This defines a new* minor network *$G' = (V', E')$ constructed as $G' = G/\{e : c_e = \top\}$, i.e., by contracting all edges with $c_e = \top$ and removing all self-loops. Vertices V' of G' are called supernodes, and we identify supernodes with the subset of nodes from V it consists of, i.e., if $s \in V'$ then $s \subseteq V$.*
- ***Consensus step.*** *Each node $v \in V$ chooses a $\tilde{O}(1)$-bit value x_v. For each supernode $s \in V'$, we define $y_s := \bigoplus_{v \in s} x_v$, where $\bigoplus$ is any pre-defined admissible aggregation operator. All nodes $v \in s$ learn y_s.*
- ***Aggregation step.*** *Each edge $e \in E'$, connecting supernodes $a \in V'$ and $b \in V'$, learns y_a and y_b and chooses two $\tilde{O}(1)$-bit values $z_{e,a}, z_{e,b}$ (i.e., one value for each endpoint). For each supernode $s \in V'$, we define an aggregate of its incident edges in E', namely $\bigotimes_{e \in \mathit{incidentEdges(s)}} z_{e,s}$ where $\bigotimes$ is some pre-defined admissible aggregation operator. All nodes $v \in s$ learn the aggregate value. To be precise, they learn the same aggregate value, a non-trivial assertion if there are many valid aggregates.*

Note that the first steps resemble the part-wise aggregation problem we sketched above. This extended definition also defines a way to exchange messages between parts through the aggregation step. Amazingly, many complex CONGEST algorithms can be broken down into part-wise aggregations. Instead of devising a long and complex algorithm, they heavily use part-wise aggregation as a black box. In each step, the algorithm either executes a *normal* CONGEST round or solves a part-wise aggregation problem. Thus, the runtime only depends on the number of part-wise aggregations and the time it takes to solve each of them. Now the parameter $\mathcal{Q}(G)$ comes into play as it is exactly the time that is needed solve the part-wise aggregation problem in G when using the CONGEST model. Most importantly for this work, the part-wise aggregation problem can be solved quickly in planar graphs in CONGEST as for these graphs, it holds $\mathcal{Q}(G) \in \widetilde{O}(\mathrm{HD})$. It is important to note that while HD may be large in certain graphs, i.e., up to n on a line graph, it is the best bound we can hope for. Furthemore, part-wise aggregation can be solved extremely fast in $\tilde{O}(1)$ in depth the PRAM model and in $\tilde{O}(1)$ in time the HYBRID model. Thus, the model does not only provide with means to develop fast algorithms in CONGEST but also in the PRAM and the HYBRID model. More precisely, it holds:

Lemma 7.1 (Implementation of the MinAgg model in CONGEST and HYBRID). *Let $\mathcal{A}$ be a τ-round* MinAgg *algorithm on a graph G as defined in Def. 7.2. Then, it holds:*

- *$\mathcal{A}$ can be implemented in $\tilde{O}(\tau \cdot \mathcal{Q}(G))$ time, w.h.p., in the CONGEST model where $\mathcal{Q}(G)$ is shortcut quality.*

  We have $\mathcal{Q}(G) \in \tilde{O}(HD + \sqrt{n})$ always, but if G does not contain K_r as a minor, it holds $\mathcal{Q}(G) \in \tilde{O}(r \cdot HD)$.
- *$\mathcal{A}$ can be implemented in $\tilde{O}(\tau)$ time, w.h.p., in the HYBRID model with $\lambda \in O(\log n)$ and $\gamma \in O(\log^2(n))$.*

Proof. The CONGEST parts were proven in [HWZ21] (for general graphs) and [GH21] (for K_r-free graphs). For the detailed proof, we refer to these papers. For the HYBRID model, we execute our algorithm from Theorem 2 on each supernode to build a well-formed tree. In a well-formed tree, all steps can be implemented in $O(\log n)$ time as it has a diamter of $O(\log n)$. In order to bring the global complexity from $O(\log^3 n)$ to the desired $O(\log^2 n)$, we need to slightly modify the algorithm. Recall that each node needs to send $O(\log^2 n)$ messages of size $O(\log n)$ in each step. Instead of sending all these messages at once, each node locally creates $O(\log n)$ batches of $O(\log n)$ messages. In each round, a node sends one batch of messages. A straightforward generalization of Lemmas 2.15 and 2.16 imply that each node only receives $O(\log n)$ messages, w.h.p. Thus, after $O(\log n)$ round, all batches are sent. Therefore, one round of the original protocol can be simulated in $O(\log n)$ rounds. Thus, after $O(\log^2 n)$ rounds, we have a well-formed tree of depth $O(\log n)$ for each supernode, w.h.p. Alternatively, we could also use the algorithm of [GHSS17] or [GHS19b] to obtain similar runtimes. Given a well-formed tree that spans the supernode, we can compute $\bigoplus$ in each supernode using this tree. Thus, we can implement the *consensus step* in time $\tilde{O}(1)$. For the *aggregation step*, each node $v \in V$ in supernode $s \in V'$ first computes $z_{(v,w),s}$ for each neighbor $w \in V$ in a different supernode. For this, it can use its local CONGEST edges as w is a neighbor in G, and the values that need to be exchanged are small. Then, it locally computes $z_v = \bigotimes_{w:w \notin s} z_{(v,w),s}$. Finally, we use well-formed tree again to compute $\bigotimes_{v \in s} z_v$ in time $\tilde{O}(1)$. □

7.1.2 Approximate Set-Source Shortest Paths in CONGEST and HYBRID

This section considers the computation of approximate shortest paths in CONGEST and HYBRID. In a distributed system, computing the shortest path means that each node learns its distance to the source and its predecessor on its path to the source, i.e., it marks one of its neighbors as a parent in an SSSP tree. Repeatedly following these parent pointers leads to the source. More precisely, we will analyze the complexity to either perform approximate SSSP from a *virtual* node that is not part of the input graph or from a subset of nodes instead of a single source and in disjoint subgraphs in parallel. As it turns out, most approximate SSSP algorithms support this variation without a (significant) loss in approximation ratio or runtime. We formalize the class of SetSSP algorithms we are interested in as follows:

Definition 7.3 (Approximate SetSSP with Virtual Nodes). *Let $G := (V, E)$ be a weighted graph and let $G' := (V', E', w)$ with $V' := V \cup \{s_1, s_2, \ldots\}$ be the graph that results from adding $\tilde{O}(1)$ virtual nodes to G. Each virtual node can have an edge of polynomially bounded weight to every virtual and non-virtual node in G'. Finally, let $S \subset V'$ be an arbitrary subset of virtual and non-virtual nodes. Then, an algorithm that solves $(1+\epsilon)$-approximate set-source shortest path with virtual nodes computes the following:*

- *Each node $v \in V' \setminus S$ learns a predecessor $p_v \in N(v)$ on a path of length at most $(1+\epsilon)d(v, S)$ to some node in S and marks the edge $\{v, p_v\}$. Together, all the marked edges imply an approximate shortest path tree T[1] rooted in set S.*
- *Each node $v \in V'$ learns its distance $d_T(v, S) \leq (1+\epsilon)d(v, S)$ to S in tree T, i.e., its exact distance to S in T and its $(1+\epsilon)$-approximate distance to S in G'.*

Note that this extension is trivial for all SSSP algorithms in the PRAM as we can just add the additional node to the input graph and handle it like any other node. However, it does not come that easy for CONGEST or the HYBRID model. In particular, this holds because we do not restrict the degree of the virtual nodes, which may be adjacent to all nodes in G. Thus, an algorithm could, in principle, require a supernode to send and receive *distinct* messages to and from all other nodes in G. A naive simulation would take $O(n \cdot \mathrm{HD})$ time, which is clearly way beyond our runtime goals. Luckily for us, the very recent breakthrough result of Rozhon ⓡ al. [RGH+22] supports virtual nodes and can be quickly implemented in CONGEST model as it heavily relies on minor aggregations. Further, it has several very beneficial properties that allow it to be also quickly implemented in HYBRID. It holds:

Lemma 7.2 ($(1+\epsilon)$-Approximate SetSSP, see [RGH+22]). *Let $G_P := (V, E, w)$ be a weighted graph with hop-diameter HD and let $\epsilon > 0$ be a parameter. Let $S \subset V$ be a set of sources. Then, there is an algorithm A_ϵ that constructs a $(1+\epsilon)$-SetSSP forest for S.*

- *In CONGEST, $\mathcal{A}_\epsilon$ in $O(\epsilon^{-2} \cdot HD)$ time, w.h.p.*

  If graph G excludes a fixed minor K_r, $\mathcal{A}_\epsilon$ can be executed in $\tilde{O}\left(\epsilon^{-2} \cdot k \cdot HD\right)$ time, w.h.p.
- *In the PRAM model with $\tilde{O}(m)$ processors, $\mathcal{A}_\epsilon$ can be executed with $\tilde{O}\left(\epsilon^{-2} \cdot m\right)$ work and $\tilde{O}\left(1\right)$ depth.*

This immediately gives us a bound for the CONGEST model. In the remainder of this section, we will therefore show that in the HYBRID model, we can perform approximate SetSSP in polylogarithmic time. More precisely, we show how to simulate the algorithm of Rozhon ⓡ al. and leverage its useful properties, such as choosing an arbitrary subset or a virtual node as a source. The exact time bounds for the simulation depend on the parameters of the HYBRID model and the properties of the input graph. The lemmas in this section are mostly a compilation of simulation results by other researchers and folklore, nevertheless we include (sketches of) their respective proofs for the sake of completeness.

We begin with (arguably) the most important result for our work. For any graph of low arboricity, which includes trees, planar graphs, and more generally all graphs that exclude a fixed minor, we can show the following:

[1] Technically, for a set S with $|S| \geq 2$, the algorithm produces a forest with the individual tree rooted in the nodes S. However, we slightly abuse notation and refer to it as a forest.

Lemma 7.3. *For any graph $G := (V, E, w)$ of bounded arboricity $\alpha_G \in \widetilde{O}(1)$, an $(1 + \epsilon)$-approximation of* SetSSP *can be computed in $\widetilde{O}(\epsilon^{-2})$ time in the HYBRID model with local of communication of $\lambda \in O(\log n)$ and logarithmic global communication $\gamma \in O(\log^2 n)$, w.h.p.*

If the input graph has low arboricity, then the HYBRID model is essentially as powerful as the PRAM. Feldmann, Hinnenthal, and Scheideler already noted this in their research on shortest path algorithms for restricted graph classes [FHS20]. For the sake of completeness, we will present their simulation result in its entrity. In the PRAM model, we assume computations are executed on a machine with p processors. The processes work over a set of shared memory cells M. The input graph G is stored in these cells. In a single step of computation, they can read and write from each cell in an atomic operation. If more than one processor writes in a cell, an arbitrary processor succeeds, i.e., we consider a CRCW PRAM. The *work* of a PRAM algorithm is the total number of primitive read or write operations that the processors perform. Further, the *depth* is the length of the longest series of operations that have to be performed sequentially. Given these defintions, we can show the following:

Lemma 7.4 (Simulating PRAM in HYBRID). *Let G be a graph with arboricity a and let $\mathcal{A}$ be a PRAM algorithm that solves a graph problem on G using N processors with depth T. Then, $\mathcal{A}$ can be simulated in time $O(a/(\log n) + T \cdot (N/n + \log n))$ in the HYBRID model with local communication of $\lambda \in O(\log n)$ and logarithmic global communication $\gamma \in O(\log^2 n)$, w.h.p.*

Proof (Verbatim from [FHS20]). Since in a PRAM the processes work over a set of shared memory cells M, we first need to map all of these cells uniformly onto the nodes. The total number of memory cells $|M|$ is arbitrary but polynomial and each memory cell is identified by a unique address x and is mapped to a node $h(x)$, where $h : M \to V$ is a pseudo-random hash function. For this, we need shared randomness. It suffices to have $\Theta(\log n)$-independence, for which only $\Theta(\log^2 n)$ bits suffice. Broadcasting these $\Theta(\log^2 n)$ bits to all nodes takes time $O(\log n)$.

To deliver x to $h(x)$, the nodes compute an $O(a)$-orientation in time $O(\log n)$ [AGG+19b]. Note that each edge in G can be represented by a constant amount of memory cells. When the edge $\{v, w\}$ that corresponds to v's memory cell with address x is directed towards v, v fills in the part of the input that corresponds to $\{v, w\}$ by sending messages to all nodes that hold the corresponding memory cells (of which there can only be constantly many). Since each node has to send at most $O(a)$ messages, it can send them out in time $O(a/\log n)$ by sending them in batches of size $\lceil \log n \rceil$.

We are now able to describe the simulation of $\mathcal{A}$: Let $k = n\lceil \log n \rceil$. Each step of $\mathcal{A}$ is divided into $\lceil N/k \rceil$ sub-steps, where in sub-step t the processors $(t-1)k + 1, (t-1)k + 2, \ldots, \min\{N, tk\}$ are active. Each node simulates $O(\log n)$ processors. Specifically, node i simulates the processors $(t-1)k + (i-1)\lceil \log n \rceil + 1$ to $\min\{N, (t-1)k + i\lceil \log n \rceil\}$. When a processor attempts to access memory cell x in some sub-step, the node that simulates it sends a message to the node $h(x)$, which returns the requested data in the next round. Since each node simulates $O(\log n)$ processors, each node only sends $O(\log n)$ requests in each sub-step. Also, in each sub-step at most $n\lceil \log n \rceil$ requests to distinct memory cells are sent in total as at most $n\lceil \log n \rceil$ are active in each sub-step. These requests are stored at positions chosen uniformly and independently at random, so each node only has to respond to $O(\log n)$ requests, w.h.p.

In a CRCW PRAM algorithm, it may happen that the same cell is read or written by multiple processors. Thus, the processors cannot send requests directly, but need to participate in aggregations towards the respective memory cells using techniques from [AGG$^+$19b]. In the case of a write, the aggregation determines which value is actually written; in the case of a read, the aggregation is used to construct a multicast tree which is used to inform all nodes that are interested in the particular memory cell about its value. Since there can be only $O(n \log n)$ members of aggregation/multicast groups, and by the argument above each node only participates and is the target of $O(\log n)$ aggregations (at most one for each processor it simulates), performing a sub-step takes time $O(\log n)$, w.h.p., by [AGG$^+$19b]. Thus, each step can be performed in time $O(N/n + \log n)$, w.h.p. (note that the additional $\log n$-overhead stems from the fact in case $N > n$, one single node still needs time $O(\log n)$ to simulate a sub-step). □

This immediately implies the following.

Corollary 7.1. *For any graph $G := (V, E, w)$ that excludes a fixed minor K_r, an $(1 + \epsilon)$-approximation of SetSSP can be computed in $\widetilde{O}(\epsilon^{-2})$ time in the HYBRID model with local of communication of $\lambda \in O(\log n)$ and logarithmic global communication $\gamma \in O(\log^2 n)$, w.h.p.*

Proof. It suffices to show that K_r-free graphs have arboricity $O(r)$. This follows from two facts. First, Nash-Williams proved that any G where every subgraph has an average degree of α has an arboricity of $O(\alpha)$ [NW64]. Second, it is known that each K_r-free graph with n nodes has at most $O(r \cdot n)$ edges [AKS23]. As any subgraph of K_r-free graph is K_r-free graph itself, it follows that every subgraph of $n' \leq n$ nodes has at most $O(r \cdot n')$ edges and, therefore, an average degree of $O(r)$. Together with the first fact, any K_r-free graph has an arboricity of $O(r)$ as claimed □

Next, we shift our attention to general graphs. If we consider general graphs of arbitrary arboricity, the runtime bound is slightly weaker as we either require massively more local communication. For unlimited local communication, Schneider [Sch23a] showed that it holds:

Lemma 7.5. *For any graph $G := (V, E, w)$, a $(1 + \epsilon)$-approximation of SetSSP can be computed in $\widetilde{O}(\epsilon^{-2})$ time in the HYBRID model with unlimited local communication.*

Schneider proves that by using LOCAL model as a basis for the local communication, one can solve approximate SetSSP as fast as in the PRAM for any graph G. He also uses the algorithm of Rozhon ⓡ al. and demonstrates how it can be efficiently simulated in the HYBRID model. The proof requires taking a closer look at the specific operations of the algorithm. In particular, Schneider [Sch23a] shows that most operations, namely all aggregations on minors, can be simulated in $O(\log n)$ time using the global communication mode of the HYBRID model and using the PRAM simulation of Lemma 7.4. However, the algorithm also requires computing a so-called Eulerian orientation of (a subgraph of) G. Here, all edges need to be assigned an orientation, s.t., they form an Euler tour of the graph that visits all nodes. On a general graph, this can not trivially be done via global communication as dense subgraphs of high arboricity could exist. To overcome this, Schneider [Sch23a] presents a LOCAL algorithm that locally sparsifies the graphs. For details on this algorithm, which relies on low-diameter decompositions, we refer the interested reader to [Sch23a]. Notably, the algorithm requires each node to (potentially) learn its $O(\log(n))$-neighborhood and can therefore be executed in $O(\log n)$

times in the LOCAL model. After executing this algorithm, each node has at most $O(\log n)$ undirected edges, which can then be oriented by another PRAM simulation using the global communication. All in all, Schneider [Sch23a] shows that all steps in the approximate SetSSP algorithm of Rozhon ⓡ al can be simulated in $\widetilde{O}(1)$ time, which implies the lemma.

Combining all of our HYBRID results gives us the following lemma:

Lemma 7.6 ($(1+\epsilon)$-Approximate SetSSP in HYBRID). *Let $G := (V, E, w)$ be a weighted graph and let $\epsilon > 0$ be a parameter. Let $S \subset V$ be a set of sources. Then, there is an algorithm A_ϵ that constructs a $(1 + \epsilon)$-SetSSP forest for S in the HYBRID model.*

- *On any graph G, $\mathcal{A}_\epsilon$ in $O(\epsilon^{-2} \cdot HD)$ time, w.h.p, with local of communication of $\lambda \in O(n)$ and logarithmic global communication $\gamma \in O(\log^2 n)$, w.h.p.*
- *If graph G has arborticity α, $\mathcal{A}_\epsilon$ can be executed in $\tilde{O}\left(\epsilon^{-2} \cdot \alpha\right)$ time, w.h.p, with local of communication of $\lambda \in O(\log n)$ and logarithmic global communication $\gamma \in O(\log^2 n)$, w.h.p.*
- *If graph G excludes a fixed minor K_r, $\mathcal{A}_\epsilon$ can be executed in $\tilde{O}\left(\epsilon^{-2} \cdot r\right)$ time, w.h.p,with local of communication of $\lambda \in O(\log n)$ and logarithmic global communication $\gamma \in O(\log^2 n)$, w.h.p.*

7.1.3 A Divide-And-Conquer Theorem for Restricted Graphs

Finally, we can combine our findings and present a meta-model for distributed algorithms consisting of shortest-path computations and aggregations. In particular, we obtain the following technical theorem:

Theorem 5. *A Divide-And-Conquer Theorem for Distributed Algorithms*

Let $G := (V, E, w)$ be a (weighted) graph and let $\mathcal{A}$ be an algorithm that can be broken down into τ synchronous steps. Each step $t \in [0, \tau]$ is of the following two types of operations:

1. A $(1+\epsilon)$-approximate SetSSP given in Def. 7.3 on a set of node-disjoint, connected subgraphs $C_1^t, \dots, C_{N_t}^t$ with $C_i^t = (V_i^t, E_i^t)$ from a set of sources $S_1^t, \dots, S_{N_t}^t$ with $S_i^t \subset V_i^t$.
2. A round of MinAgg given in Def. 7.2 on a set of node-disjoint, connected subgraphs $C_1^t, \dots, C_{N_t}^t$ with $C_i = (V_i, E_i)$ as the supernodes described in the consensus step.

Note that the subgraphs $C_1^t, \dots, C_{N_t}^t$ can be different in each step.

Suppose that $\mathcal{A}$ can be broken down into τ_s steps of $(1+\epsilon)$-approximate SetSSP computations and τ_m minor aggregations. Then, it holds:

- In CONGEST, $\mathcal{A}$ can be executed in $\tilde{O}\left((\epsilon^{-2} \cdot \tau_s + \tau_m) \cdot (\mathrm{HD} + \sqrt{n})\right)$ time, w.h.p.

  If graph G excludes a fixed minor K_r, $\mathcal{A}$ can be executed in $\tilde{O}\left((\epsilon^{-2} \cdot \tau_s + \tau_m) \cdot r \cdot \mathrm{HD}\right)$ time, w.h.p.
- In the HYBRID model, $\mathcal{A}$ can be executed in $\tilde{O}\left((\epsilon^{-2} \cdot \tau_s + \tau_m)\right)$ time, w.h.p., with local of communication of $\lambda \in O(n)$ and logarithmic global communication $\gamma \in O(\log^2 n)$, w.h.p.

  If graph G excludes K_r as a minor, $\mathcal{A}$ can be executed in $\tilde{O}\left((\epsilon^{-2} \cdot \tau_s + \tau_m) \cdot r\right)$ time, w.h.p., with local of communication of $\lambda \in O(\log n)$ and logarithmic global communication $\gamma \in O(\log^2 n)$, w.h.p.

Proof. The theorem follows from a straightforward combination of Lemmas 7.1, 7.2, and 7.6. In the following, we fix a single step t and the corresponding set of node-disjoint, connected subgraphs $C_1^t, \dots, C_{N_t}^t$. We omit t in the sub- and superscript for an easier notation as it stays fixed. Further, define

$$E_{\mathcal{C}} := \{\{v, w\} \in E \mid v \in C_i, w \in C_j, i \neq j\} \tag{7.1}$$

to be the set of edges between nodes of different subgraphs.

First, suppose that in step t, we perform a round of MinAgg on $C_1, \dots, C_N$ as given in Def. 7.2. In other, in the consensus step, we contract all edges in $E_{\mathcal{C}}$, so that $C_1, \dots, C_N$ are the resulting supernodes/minors. Following Lemma 7.1, a round of MinAgg can be performed in $\tilde{O}(\mathrm{HD} + \sqrt{n})$ time, w.h.p., in general graphs and $\tilde{O}(r \cdot \mathrm{HD})$ time, w.h.p. For HYBRID, we obtain a runtime of $\tilde{O}(1)$. Thus, the total time complexity of all τ_m aggregation steps, w.h.p., is $\tilde{O}(\tau_m \cdot \mathrm{HD})$ in CONGEST on any graph, $\tilde{O}(\tau_m \cdot r \cdot \mathrm{HD})$ in CONGEST on a K_r-free graph, and $\tilde{O}(\tau_m)$ in the HYBRID model.

Next, suppose that t is a shortest path step. Here, we need to do a bit more work. First, we aggregate the length of the longest edge in the graph. This can be done with a single minor aggregation as given in Def. 7.2, where each node picks its longest incident edge as input, and we aggregate the maximum. We denote this length as W. With knowledge of W, can effectively assign *infinite* weight to all edges between components.

More precisely, we choose a weight of $2Wn$ where W is the highest weight in the graph. This yields the weight function

$$w'(e) := \begin{cases} 2Wn & \text{if } e \in E_{\mathcal{C}} \\ w(e) & \text{else.} \end{cases} \tag{7.2}$$

We denote the resulting graph as $G' = (V, E, w')$. Note that we only change the weights but not the graph's topology.

Finally, perform a SetSSP with $\epsilon < 1$ on G from $S_1 \cup \ldots \cup S_N$, i.e., the union of all source sets of all components. In the resulting SetSSP tree $\mathcal{T}$, each node $v \in C_i$ will always be placed in the subtree of its component's source set S_i. Since the paths to all sources of other components must contain an edge of length $2Wn$, the leader of the $v's$ component must always be the closest if the path is at least 2- approximate.

Thus, each shortest path step consists of a minor aggregation whose runtime we already bounded and one SetSSP computation on a connected graph G' that has the same topology as G. For the CONGEST and the HYBRID model, the proclaimed bounds now follow directly from Lemmas 7.2 and 7.6. $\square$

This theorem states a divide-and-conquer algorithm that recursively divides input graphs into disjoint subgraphs (and is then applied to these subgraphs) can be efficiently implemented in CONGEST and HYBRID. If the graph is K_r-free graphs, the runtimes are very competitive in both models. We will heavily use this theorem in the descriptions of the algorithms in this part. In particular, instead of giving exact pseudocodes for the implementation in CONGEST and HYBRID, we will break the algorithms down to minor aggregation and shortest path computations in each subgraph. Theorem 5 then allows us to derive the runtime for the algorithm in CONGEST and HYBRID.

Finally, we remark that, our class of algorithm also performs well on k-path separable graphs. For the proof, we first consider useful properties of k-path separable graphs. The first important fact is that the class of universally k-path separable graphs is closed under minor taking. That means it holds:

Lemma 7.7 (Theorem 2 in [DG10]). *Let $G = (V, E, w)$ be universally k-path separable graph, then any minor H of G is also universally k-path separable.*

The proof can be found in [DG10] and exploits the fact that the graphs are separable for *all* weight functions. Further, it was shown in [DG10] that a universally k-path separable graph does not contain K_{4k+1}, the clique of $4k + 1$ nodes, as a minor. The proof for this is straightforward. Clearly, K_{4k+1} is not universally k-path separable. Suppose all edges have weight 1; then every edge is the shortest path between its endpoints. After removing *any* set of k edges, the remaining $2k + 1 > (1/2) \cdot (4k + 1)$ nodes are still connected as we consider a clique. Therefore, there can be no separator consisting of only k shortest paths and K_{4k+1} is not universally k-path separable. As the class of universal k-path separators is closed under minor taking, it cannot contain K_{4k+1}. Thus, it holds:

Lemma 7.8 (Corollary of Theorem 2 and Proposition 1 in [DG10]). *Let $G = (V, E, w)$ be universally k-path separable graph, then does not contain K_{4k+1} as a minor.*

This fact is important as this fact is strongly related to the runtime of our algorithms in CONGEST and HYBRID due to Theorem 5. Because each universally k-path separable path excludes K_{4k+1} as a minor, Theorem 7.1.3 immediately implies the following.

Lemma 7.9. *Let $G := (V, E, w)$ be a (weighted) universally k-path separable graph and let $\mathcal{C}_1, \dots, \mathcal{C}_N$ be set of disjoint subgraphs of G. Further, let $\mathcal{A}$ be an algorithm that is independently executed on each $\mathcal{C}_1, \dots, \mathcal{C}_N$. Suppose that $\mathcal{A}$ can be broken down into τ_s steps of $(1+\epsilon)$-approximate SetSSP computations with $\epsilon < 1$ as defined in Definition 7.3 and τ_m minor aggregations as defined in Definition 7.2. Then, $\mathcal{A}$ can be executed on all $\mathcal{C}_1, \dots, \mathcal{C}_N$ in parallel in $\tilde{O}\left((\epsilon^{-2} \cdot \tau_s + \tau_m) \cdot k \cdot HD\right)$ time in CONGEST and $\tilde{O}\left(\epsilon^{-2} \cdot \tau_s + \tau_m\right)$ time in HYBRID with local of communication of $\lambda \in O(\log n)$ and logarithmic global communication $\gamma \in O(\log^2 n)$, w.h.p.*

7.2 The Tree Operations of Ghaffari and Zuzic

Throughout our algorithms, we will often need to work on a forest, i.e., sets of disjoint trees. We use the following lemma for recurring tasks on these trees:

Lemma 7.10 (Tree Operations, Based on [GZ22a]). *Let $F := (T_1, \dots, T_m)$ be a subforest (each edge e knows whether $e \in E(F)$ or not) of a planar graph and suppose that each tree T_i has a unique root $r_i \in V$, i.e., each node knows whether it is the root and which of its neighbors are parent or children, if any. Now consider the following three computational tasks:*

1. ***AncestorSum and SubtreeSum:** Suppose each node $v \in T_i$ has an $\widetilde{O}(1)$-bit private input x_v. Further, let $Anc(v)$ and $Dec(v)$ be the ancestors and descendants of v w.r.t. to r_i, including v itself. Each node computes $A(v) := \bigotimes_{w \in Anc(v)} x_w$ and $D(v) := \bigotimes_{w \in Dec(v)} x_w$.*

2. ***Path Selection:** Given a node $w \in T_i$, each node $v \in T_i$ learns whether it is on the unique path from r_i to w in T_i.*

3. ***Depth First Search Labels:** Each node $v \in T_i$ computes its unique entry and exit label of a depth first search started in r_i.*

All of these tasks can be implemented in one round of minor aggregation.

Proof. The proofs are straightforward.

1. **Ancestor and Subtree Sum:** This was directly shown in [GZ22a, Lemma 16].

2. **Path Selection:** We perform a single minor aggregation with $x_w = 1$ and $x_v = 0$ for $v \in T_i \backslash \{w\}$ where we contract the unique path from w to r_i performing no actions in the Consensus step. Every node with value 1 then marks itself a part of the path.

3. **Depth First Search Labels:** We perform Ancestor sum and Subtree sum to count the number of nodes on each node's root path and subtree. Each node informs its parent about its subtree size. We order the children by ascending subtree size, breaking ties arbitrarily. To obtain the labels of one of its child nodes, a parent combines the length of its root path with the sizes of the subtrees traversed before that node.

□

8 Weak Separators Via Approximate Distances

THIS section considers the distributed computation of separators for restricted graphs, notably for the general class of so-called k-path separable graphs. Recall that a separator S is a *subgraph* of G such that in the induced subgraph $G \setminus S$ that results from removing S from G, every connected component only contains a constant fraction of G's nodes. The contents of this chapter are lifted from the following publication:

> Jinfeng Dou, Thorsten Götte, Henning Hillebrandt, Christian Scheideler, and Julian Werthmann. Distributed and parallel low-diameter decompositions for arbitrary and restricted graphs. In Raghu Meka, editor, *16th Innovations in Theoretical Computer Science Conference (ITCS 2025), New York City, NY, USA, January 7–10, 2025*, Leibniz International Proceedings in Informatics (LIPIcs), page (to appear), Dagstuhl, Germany, 2025. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik

Separators are a central tool in designing algorithms for restricted graph classes as they give rise to efficient divide-and-conquer algorithms. The general idea is to compute a separator in each recursive step, remove it, and then recurse on the emerging components. Our algorithms for computing clusterings and routing schemes are no exception to this, as they will follow this generic pattern. Naturally, this scheme implies a recursion depth of only $O(\log n)$. Thus, there is a great interest in the efficient distributed construction of separators as their computing usually dominates the runtime of a single recursive step. As our main technical contribution, we present the fast construction of a weaker form of separators, which we will call $\mathcal{D}$-weak separators. A $\mathcal{D}$-weak separator S' only ensures that in the graph $G \setminus S$, all nodes have at most a constant fraction of nodes in distance

$\mathcal{D}$. In particular, in contrast to a *classical* separator, the graph $G \setminus S'$ might still be connected. Formally, we define them as follows:

Definition 8.1 (Weak κ-Path $(\mathcal{D},\epsilon)$-Separator). *Let $G := (V,E,w)$ be a weighted graph, $\mathcal{D} > 1$ be an arbitrary distance parameter, and $\epsilon > 0$ be an approximation parameter. Then, we call the set $S(\mathcal{D},\epsilon) := (\mathcal{P}_1,\dots,\mathcal{P}_\kappa)$ with $\mathcal{P}_i := (P_i, B_i)$ a weak κ-path separator, if it holds:*

1. *Each $P_i \in \mathcal{P}_i$ is a (approximate) shortest path in $G \setminus \bigcup_{j=1}^{i-1} \mathcal{P}_j$ of length at most $4\mathcal{D}$.*
2. *Each $B_i \subseteq B_G(P_i, \epsilon\mathcal{D})$ is a set of nodes surrounding path P_i.*
3. *For all $v \in (V \setminus \bigcup_{j=1}^{\kappa} \mathcal{P}_i)$ it holds $|B_{G\setminus S}(v,\mathcal{D})| \leq (7/8) \cdot n$.*

Note that this definition is generic enough also to include vanilla k-path separators (where each set B_i only contains the path itself and no further nodes) and *traditional* vertex separators (where each path is a single node). For general k-path separable graphs, we prove the following theorem:

Theorem 6. *Weak Separators for k-Path Separable Graphs*
Consider a weighted k-path separable graph $G := (V,E,w)$ with weighted diameter smaller than $\mathcal{D}$. For $\epsilon \geq 0$, there is an algorithm that constructs a weak $O(\epsilon^{-1} \cdot k \cdot \log n)$-path $(\mathcal{D},\epsilon)$-separator, w.h.p. The algorithm uses $\tilde{O}(\epsilon^{-1} \cdot k \cdot \log n)$ minor aggregations and $O(\epsilon^{-1} \cdot k \cdot \log n)$ 2-approximate SetSSP computations, w.h.p.

Recall that k-separable graphs are K_{4k+1}-free and therefore minor aggregations and SetSSP can be implemented extremly efficiently in both CONGEST and the HYBRID model. Thus, this theorem implies that the separators can be computed in $\tilde{O}(\epsilon^{-3} \cdot k^2 \cdot \mathrm{HD})$ time, w.h.p., in CONGEST. Further, in the HYBRID model, the separator can be computed in $\tilde{O}(\epsilon^{-2} \cdot k)$ time, w.h.p. Assuming that $k \in \tilde{O}(1)$ and $\epsilon^{-1} \in \tilde{O}(1)$, the runtimes simplify to $\tilde{O}(\mathrm{HD})$ and $\tilde{O}(1)$ respectively. In all case, we assume that the HYBRID model has local capacity of $\lambda \in O(\log n)$ and a global capacity of $O(\log^2 n)$. All these runtimes follow directly from Lemma 7.9.

The remainder of this section is structured as follows: First, we elaborate on why we compute weak separators instead of proper separators. Then, in Section 8.2, we present the algorithm behind Theorem 6. Finally, in Section 8.3, we prove the algorithm's correctness and runtime. The main ingredient of our proof is a structural property of k-path separators that allows us to sample them efficiently.

8.1 Why Weak Separators?

Having defined k-path separable graphs, which by definition contain a proper k-path separator, the obvious question is why we settle for a *weak* separator instead. To this end, let us review prior results and identify possible difficulties. Starting with positive results, it *is* possible to construct path separators for planar graphs in a distributed setting. In fact, the state-of-the-art algorithms for computing DFS trees in [GP17] or computing a graph's (weighted) diameter in [LP19] in planar graphs construct a path separator for planar graphs. The algorithm present in these papers constructs a path separator in $\tilde{O}(\mathrm{HD})$ time, w.h.p. In the next chapter, we show how to extend this algorithm to be executed in a series of *arbitrary* connected subgraphs.

However, while the existence of an efficient algorithm for a planar graph gives hope, it turns out to be very tough for other k-path separable graphs. The constructions in both [GP17] and [LP19] heavily exploit the fact that a (planar) embedding of a (planar) graph can be computed in $\tilde{O}(\mathrm{HD})$ time in CONGEST due to Ghaffari and Haeupler in [GH16b]. Given a suitable embedding, the ideas could be generalized. Thus, finding a suitable embedding could be a way to extend this algorithm to more general k-separable graphs. A good candidate for such an embedding can be found in [AG06]. Abraham and Gavoille present an algorithm that computes a $f(r)$-path separator for K_r-free graphs [AG06]. Their algorithm relies on a complex graph embedding algorithm by Seymour and Robertson[RS86]. On a very high level, the embedding divides G into subgraphs that can almost be embedded on a surface similar to planar graph. By *almost*, we means that in each subgraph, there is only small number of non-embeddable parts (with special properties) that need to be handled separately. The concrete number of these parts depends only on r. Given such an embedding, Abraham and Gavoille show that one can efficiently compute k-path separator.

Sadly, we have little hope that the algorithm that computes the embedding can be efficiently implemented in a distributed (or even parallel) context. It already requires $n^{O(r)}$ time in the sequential setting. Moreover, it requires sophisticated techniques that can not be trivially sped up by running them in parallel. Thus, it remains elusive (to the author at least) how to implement this algorithm in any distributed or even parallel model. Therefore, we settle for the weaker notion of separator simply because they easier to compute and suffice for our purposes.

8.2 Algorithm Description

Having established why it is difficult to compute k-path separators, we can proceed to the initially promised weak separators. More precisely, we present the algorithm behind Theorem 6. Namely, for a given k-path separable graph $G := (V, E, w)$, we give an algorithm constructs a weak $O(\epsilon^{-1} \cdot k \cdot \log n)$-path $(\mathcal{D}, \epsilon)$-separator for parameters $\mathcal{D}$ and ϵ. Our main insight is that instead of computing an embedding in a distributed and parallel setting, we show that we get very similar guarantees using a weaker separator that can be computed without the embedding. Thus, we take a different approach to completely avoid the (potentially expensive) computation of an embedding. We strongly remark that, in doing so, we sacrifice some of the separator's useful properties. However, in our applications, this only results in some additional polylogarithmic factors in the runtime.

Surprisingly, the algorithm behind the Theorem can be stated in just a few sentences. The core idea is to iteratively sample approximate shortest paths and nodes close to these paths and remove them from the graph until all remaining nodes have few nodes in distance $\mathcal{D}$, w. h.p. More precisely, we start with a graph $G_0 = G$. The algorithm proceeds in $T \in O(\epsilon^{-1} \cdot k \cdot \log n)$ steps where in step t, we consider graph $(V_t, E_t) := G_t \subset G_{t-1}$. A single step t works as follows: First, we pick a node $v \in V_t$ uniformly at random. Then, we compute a 2-approximate shortest path tree T_v from w. Let $W_v \subset V_t$ be set of all nodes in distance at most $4 \cdot \mathcal{D}$ to v in

T_v. We pick a node $w \in W_v$ uniformly at random and consider the corresponding path $P_t = (v, \dots, w)$. Using P_t as a set-source, we compute a 2-approximate shortest path tree T_{P_t} from P_t. Now denote B_{P_t} to be the nodes in the distance at most $\epsilon \cdot \mathcal{D}$ to P_t in G (and **not** G_t). Finally, we remove P_t and B_{P_t} from G_t to obtain G_{t+1}. Note that, due to the approximation guarantee, this removes all nodes in distance at least $\epsilon/2 \cdot \mathcal{D}$ in G. Further, all steps can be computed using 2-approximate paths (and a few aggregations to pick the random nodes) and completely devoid of complex embedding.

More formally, the algorithm consists of $T \in O(\epsilon^{-1} \cdot k \log n)$ sequential steps. In each step $t \leq T$, we construct a path P_t and ball $K_t \supset P_t$ around that path. Further, we define $G_t := (V_t, E_t, w)$ to be the graph induced by all nodes that have not yet been added to any set $B_1, \dots, B_t$. Finally, we fix the parameter $\epsilon' := \epsilon/2$. Given these definitions, a single step of the algorithm consists of the following five operations:

(Step 1) Choose a random node $v_t \in V_{t-1}$: Pick a node $v \in V_{t-1}$ uniformly at random from all active nodes.

(Step 2) Construct a 2-approximate SSSP tree T_{v_t} for v_t: Perform a 2-approximate SSSP from node v_t. Afterwards, all nodes $w \in V$ know a value $d_{T_{v_t}}(w, v_t)$, which is its 2-approximate distance to v_t in graph G_{t-1}. It holds for all $w \in V_{t-1}$:

$$d_{G_{t-1}}(v_t, w) \leq d_{T_{v_t}}(v_t, w) \leq 2d_{G_{t-1}}(v_t, w)$$

(Step 3) Choose a random path P_t: Given the distances computed in the previous step, each node checks whether it is in the distance at most $4\mathcal{D}$ to v_t. In particular, we define the set

$$N_{v_t} := \left\{ w \in V_{t-1} \mid d_{T_{v_t}}(v_t, w) \leq 4\mathcal{D} \right\}$$

Sample one of these nodes uniformly at random.

(Step 4) Compute an Approximation of $B_G(P_t, \epsilon \cdot \mathcal{D})$: Compute a 2-approximate shortest path tree for $\{P_t\}$ in G. Thereby, we obtain a 2-approximate SSSP tree T_{P_t} and all nodes $w \in V_{t-1}$ know a value $d_{T_{P_t}}(w, P_t)$, which is its 2-approximate distance to P_t in graph G. After that, every node knows whether it is in the set

$$K_t := \left\{ w \in V_t \mid d_{T_{P_t}}(w, P_t) \leq 2\epsilon' \mathcal{D} \right\}$$

(Step 5) Remove K_t from G_{t-1}: All nodes use their previously computed distances to determine whether they are in K_t. If so, they remove themselves from the graph, setting $G_t := G_{t-1} \setminus K_t$.

Note that the nodes K_t removed in step t will not be considered for picking the path to the next iteration. However, they will be considered for computing the balls around the paths.

8.3 Analysis

In this section, we will analyze the sampling algorithm and prove Theorem 6. Clearly, for any choice of ϵ, the algorithm produces sets $(P_1, K_1), (P_1, K_1) \dots$ that fit the description of the sets $\mathcal{P}_1, \mathcal{P}_2, \dots$ in Definition 8.1. By construction, all paths $P_1, P_2, \dots$ are of bounded length $4 \cdot \mathcal{D}$ and each K_t contains all nodes bounded in distance $\epsilon \cdot \mathcal{D}$ from each path P_t. The latter follows as we chose $\epsilon' = \epsilon/2$, so each K_t is a strict subset of $B_G(P_t, \epsilon \cdot \mathcal{D})$ and a superset of $B_G(P_t, \epsilon/2 \cdot \mathcal{D})$. Therefore, to prove Theorem 6, it remains to show that for $T \in O(\epsilon^{-1} \cdot k \cdot \log n)$ removing the sets $K_1, \dots, K_T$ weakly separates the graph, i.e., for all nodes $v \in V_T$, it holds:

$$|B_{G_T}(v, \mathcal{D})| \leq (7/8) \cdot n. \tag{8.1}$$

Further, we must show that the algorithm can be implemented in $\tilde{O}(T)$ minor aggregations and 2-approximate SetSSP computations.

As our main analytical tool, we consider the following potential function Φ_t for each intermediate graph G_t that counts the number of nodes with more than $(7/8) \cdot n$ nodes in distance $2\mathcal{D}$. Formally, we have:

$$\Phi_t := \left|\left\{ v \in V_t \;\middle|\; |B_{G_t}(v, 2\mathcal{D})| \geq (7/8) \cdot n \right\}\right|$$

Further, for brevity, we write $\Phi_t(v) = 1$ if and only if $|B_{G_t}(v, 2\mathcal{D})| \geq (7/8) \cdot n$ and $\Phi_t(v) = 0$, otherwise. Note that $\Phi_t := \sum_{v \in V} \Phi_t(v)$.

We make the following crucial observation about this potential: As soon as this potential drops below $(7/8) \cdot n$, we have more than $(1/8) \cdot n$ nodes with less than $(7/8) \cdot n$ nodes in distance $2\mathcal{D}$. This implies that no node can have more than $(7/8) \cdot n$ nodes in distance $\mathcal{D}$. Suppose for contradiction that there is a node $v \in V_t$ with more than $(7/8) \cdot n$ nodes in distance $\mathcal{D}$. By the triangle inequality, all these nodes would be in distance $2\mathcal{D}$ to each other. Thus, the potential would be bigger than $(7/8) \cdot n$ as there are more than $(7/8) \cdot n$ nodes with more than $(7/8) \cdot n$ nodes in distance $\mathcal{D}$. This is a contradiction; therefore, the nodes sampled until this point are the desired weak separator. Thus, we want to bound the number of steps until the potential drops. To be precise, we prove the following lemma:

Lemma 8.1. *Let G_T be the graph after the Tth step of our algorithm. Then, for $T \in O\left(c \cdot \epsilon^{-1} \cdot k\right)$, it holds:*

$$\mathbf{Pr}[\Phi_T \leq (7/8) \cdot n] \geq 1 - O(e^{-c})$$

Thus, after $T \in O(\epsilon^{-1} \cdot k \cdot \log n)$ iterations, it holds $\Phi_T \leq (7/8) \cdot n$. This implies for all $v \in V_{\tau_c}$ that:

$$B_{G_T}(v, \mathcal{D}) \leq (7/8) \cdot n \tag{8.2}$$

Therefore, for $T \in O(\epsilon^{-1} \cdot k \cdot \log n)$, the algorithm successfully computes a weak $O(\epsilon^{-1} \cdot k \cdot \log n)$-path $(\mathcal{D}, \epsilon)$-separator, w.h.p. as desired. Before we proceed to the formal proof of the lemma, let's clarify the intuition and difficulties behind our analysis. For the sake of argument, assume that the k-path separator $S = (\mathcal{P}_1, \mathcal{P}_2, \dots)$

of G only consists if shortest paths in G itself. On a high level, we want to show that the sets $K_1, \dots, K_T$ *either* quickly cover the graph's k-path separator $S = (\mathcal{P}_1, \mathcal{P}_2, \dots)$ *or* the potential drops by constant fraction. As intuitively, many paths must cross the separator, we will likely pick such a path through random sampling. Recall that we sample paths of length at most $4\mathcal{D}$ with an area of at least $\epsilon/2\mathcal{D}$ around them. Thus, we cover a subpath of length $\epsilon/2\mathcal{D}$ whenever we pick a path that intersects with the separator. Say that a path in the separator S is of (weighted) length $\mathcal{D}'$. Then, we require at least $O(\mathcal{D}'/\epsilon\mathcal{D})$ crossings to cover it. In a k-separable graph G where all paths are shortest paths in the separator are shortest paths in G, we can bound the length of each path by G's diameter $\mathcal{D}$. Thus, after sampling $\epsilon \cdot k$ paths that crossed (a path in) the separator S, we completely removed it from G. As any superset of a separator is obviously also a separator, we are done. However, while initially it is likely to sample a path that crosses the separator, this probability steadily declines. Surprisingly, a low probability of crossing the separator is good for our purposes. A low probability of crossing the separator means that many nodes have no path (or very few) paths of length $2\mathcal{D}$ to nodes reachable only via the separator. As for each node, there are at least $(1/2) \cdot n$ nodes only reachable via the separator, this implies a low overall potential. Combining these two facts implies that after sampling $T \in (\epsilon^{-1} \cdot k)$ paths, either $\epsilon^{-1} \cdot k$ paths crossed S removing it from the graph, or the potential dropped as desired. In either case, we are done. Figure 8.1 shows a visualization of our proof's main ideas.

Sadly, we cannot directly apply this technique to all k-path separable graphs. A major factor that complicates this is the fact that, in general, the paths in a k-path separator are divided into subsets $(\mathcal{P}_1, \mathcal{P}_2, \dots)$. The paths in $\mathcal{P}_i$ with $i > 1$ are only the shortest paths if the paths from previous subsets are removed. Thus, if the (weighted) diameter of G is $\mathcal{D}$, this only gives us a bound for the paths in $\mathcal{P}_1$. This does not help us when considering the lengths of the paths in the induced graph $G \setminus \mathcal{P}_1$. The diameter of $G \setminus \mathcal{P}_1$ and, therefore, the length of the paths in $\mathcal{P}_2$ are unbounded. Now, recall that we do **not** need to cover the whole separator as it is sufficient that each node only has a constant fraction of nodes in distance $\mathcal{D}$. Our main trick is only considering a carefully picked subset of the separator paths that intersect with paths until length $O(\mathcal{D})$. This subset only consists of paths of bounded length that separate not all, but still a *large enough* fraction of the nodes. Formally, we show the following useful fact about k-path separators, namely

Lemma 8.2. *Let $G := (V, E, w)$ be a weighted k-path separable graph of n nodes with (weighted) diameter $\mathcal{D}$. Then, there exists a set $\mathcal{B} = \{P_1, \dots, P_\kappa\}$ with $\kappa \leq k$ simple paths of length $32\mathcal{D}$ such that for all $v \in V$, it holds $B_{G\setminus\mathcal{B}}(v, 4\mathcal{D}) \leq (3/4) \cdot n$.*

In other words, a subset of bounded length paths intersects with a constant fraction of all paths our algorithm can potentially sample. The proof requires copious use of the triangle inequality and the pigeonhole principle and can be found in Section 8.3.1. In the following, we provide a simplified sketch that summarizes the main ideas but omits technical details:

Proof Sketch. First, we show that there is a set $\mathcal{B} = \{P_1, \dots, P_\kappa\}$ with $\kappa \leq k$ simple paths of length $32\mathcal{D}$ such that for a subset $\mathcal{Z}$ with $|\mathcal{Z}| \geq (3/4) \cdot n$, it holds $B_{G\setminus\mathcal{B}}(v, 8\mathcal{D}) \leq (3/4) \cdot n$.

To this end, consider the graphs $(G^{(0)}, G^{(1)}, \dots)$ with $G^{(i)} := G \setminus \bigcup_{j\leq i} \mathcal{P}_i$, that result from removing the separator paths. Let $G^{(i_\mathcal{D})}$ be the first graph where **all** nodes have less than $(3/4)n$ nodes in distance $8\mathcal{D}$. This graph must exist by the pigeonhole principle as initially, all nodes have n nodes in the distance $8\mathcal{D}$, and once the whole separator is removed, there can only be $n/2$. Then, there is at least one node $z \in V$ with at least $(3/4)n$ nodes in distance $8\mathcal{D}$ in all graphs $G, \dots, G^{(i_{\mathcal{D}}-1)}$. We denote these nodes as $\mathcal{Z} := B_{G^{(i_{\mathcal{D}}-1)}}(z, 8\mathcal{D})$. Further, by the triangle inequality, all of these $(3/4) \cdot n$ nodes are in the distance $16\mathcal{D}$ to one another in all graphs $G, \dots, G^{(i_{\mathcal{D}}-1)}$. This is clearly the case in $G^{(i_{\mathcal{D}}-1)}$, and because we only remove nodes and edges, they can only be closer in earlier graphs. Next, consider a graph $G^{(i)} := G \setminus \bigcup_{j \leq i} \mathcal{P}_i$ with $1 \leq i \leq i_{\mathcal{D}} - 1$ and $\mathcal{P}_{i+1}$, the set of shortest paths in $G^{(i)}$ in the k-path separator. For each $P_j \in \mathcal{P}_{i+1}$, observe the set

$$P_j' := \bigcup_{v \in \mathcal{Z}} P_j \cap B_{G^{(i)}}(v, 8\mathcal{D})$$

Consider two nodes $s, t \in P_j'$. By definition, there are nodes $v_s, v_t \in \mathcal{Z}$ with $s \in B_{G^{(i)}}(v_s, 8\mathcal{D})$ and $t \in B_{G^{(i)}}(v_t, 8\mathcal{D})$. As $v_s, v_t \in \mathcal{Z}$ their distance in $G^{(i)}$ is at most $8\mathcal{D}$. Therefore, by the triangle inequality, two nodes in P_j' are at most in distance $32\mathcal{D}$. As P_j is a shortest path in G_i, the shortest connected subpath $\tilde{P}_j$ of P_j that contains P_j' has at most length $16\mathcal{D}$.

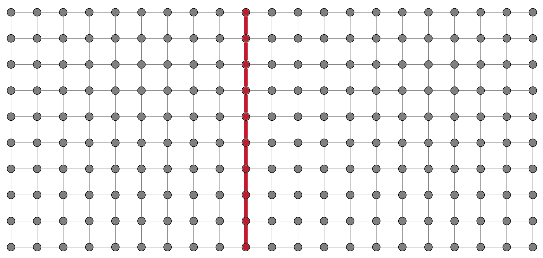
Finally, let $\tilde{\mathcal{P}}_{i+1} := (\tilde{P}_1, \tilde{P}_2, \dots)$ for $P_j \in \mathcal{P}_{i+1}$ be the set of all these subpaths for all $P_j \in \mathcal{P}_{i+1}$. Then, the union $\mathcal{B} := \{\tilde{\mathcal{P}}_1, \dots, \tilde{\mathcal{P}}_{i_{\mathcal{D}}}\}$ of all these subpaths is the desired set $\mathcal{B}$. It consists of at most k paths of length $32\mathcal{D}$. Further, a simple induction reveals that $|B_{G \setminus \mathcal{B}}(v, 8\mathcal{D})| = |B_{G^{(i_{\mathcal{D}})}}(v, 8\mathcal{D})|$ as $\mathcal{B}$ contains all nodes of $(\mathcal{P}_1, \dots, \mathcal{P}_{i_{\mathcal{D}}})$ that intersect with any $B_{G^{(i)}}(v, 8\mathcal{D})$. Recall that in $G^{(i_{\mathcal{D}})}$ all nodes have less than $(3/4)n$ nodes in distance $8\mathcal{D}$. Thus, it holds $|B_{G \setminus \mathcal{B}}(v, 8\mathcal{D})| \leq (3/4) \cdot n$.

Finally, consider any node $v \in V$. Now suppose for contradiction that v has more than $(3/4) \cdot n$ nodes in distance $4\mathcal{D}$ after removing $\mathcal{B}$. Then, $B_{G \setminus \mathcal{B}}(v, 4\mathcal{D})$ contains at least one node of $w \in \mathcal{Z}$. The triangle inequality implies $B_{G \setminus \mathcal{B}}(v, 4\mathcal{D}) \subseteq B_{G \setminus \mathcal{B}}(w, 8\mathcal{D}$. Thus, it holds $|B_{G \setminus \mathcal{B}}(w, 8\mathcal{D}| \geq |B_{G \setminus \mathcal{B}}(v, 4\mathcal{D}| > (3/4) \cdot n$. This is a contradiction as $|B_{G \setminus \mathcal{B}}(v, 4\mathcal{D}| \leq (3/4) \cdot n$. Therefore, no node $v \in V$ can have more than $(3/4) \cdot n$ nodes in distance $4\mathcal{D}$ after removing $\mathcal{B}$.

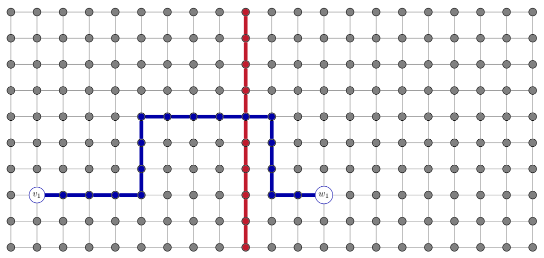
□

Equipped with this lemma, we now consider the event that the algorithm samples a path P_t that *crosses* some path of the set $\mathcal{B}$, i.e., the path we sample contains (at least) one node from one of the k paths in $\mathcal{B}$. In this case, we remove a subpath of length $\epsilon\mathcal{D}$ from this path when we remove the set B_{P_t} around the P_t. This follows because we determine the set B_t with respect to the original graph G and not G_t. Thus, after sampling $O(\epsilon^{-1} \cdot k)$ paths that cross $\mathcal{B}$, we must have completely removed $\mathcal{B}$ from G. If $\mathcal{B}$ is removed, each node has at most $(3/4) \cdot n$ nodes in distance $\mathcal{D} \leq 2\mathcal{D}$ per definition and we are done.

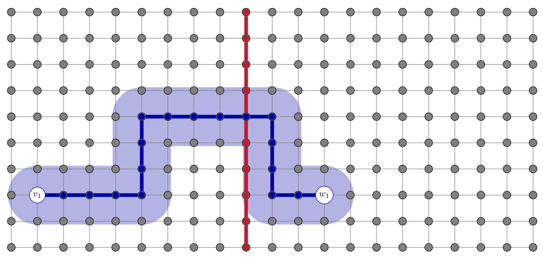
To prove our main theorem, we will bound the time until **either** the potential drops **or** $O(\epsilon^{-1} \cdot k)$ paths cross $\mathcal{B}$. In the following, we sketch the general approach of the proof and convey its main idea. We refer to Section 8.3.2 for a complete proof that includes all technical details. Starting off, we denote the event that the path from v_t to w_t crosses $\mathcal{B}$ as $v_t \underset{\mathcal{B}}{\rightsquigarrow} w_t$. By definition of $\mathcal{B}$, for each node, the set of nodes reachable via $\mathcal{B}$ is at least size $(1/4) \cdot n$. Thus, for all nodes $v \in V_t$ with $\Phi_t(v) = 1$, at least $(1/4) \cdot n - (1/8) \cdot n \geq (1/8) \cdot n$ of these nodes must still be in distance $2\mathcal{D}$. All these nodes might be chosen as the path's endpoint P_t if we pick v as the starting point as the 2-approximate distance is at most $4\mathcal{D}$. So, if we sample one of them, we cross $\mathcal{B}$ as all paths



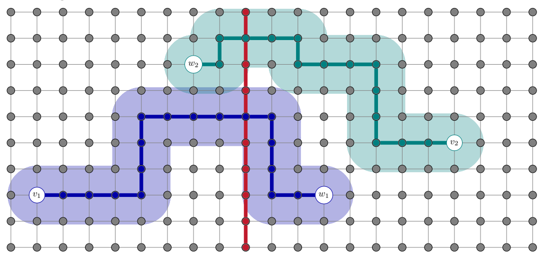
(a) Marked in red is a separator path unknown to the algorithm.



(b) When sampling a random path of bounded length, the probability of crossing the separator is constant.



(c) We remove nodes in distance $\epsilon\mathbf{D}$ around the path. This also removes an $\epsilon\mathcal{D}$-fraction of the separator.



(d) If the residual graph is not well-separated, the probability of crossing (what remains of) the separator is still constant.

Figure 8.1: We choose a two-dimensional mesh for illustration. While the operations in Figures **(b)** and **(c)** are straightforward to prove, the core of our analysis we will be the claim we make in Figure **(d)**.

of length at most $4\mathcal{D}$ cross $\mathcal{B}$. Thus, in a configuration with potential $\Phi_t \geq (7/8) \cdot n$, the probability sample a path that crosses $\mathcal{B}$ is at least

$$\mathbf{Pr}\left[v_t \underset{\mathcal{B}}{\rightsquigarrow} w_t \mid \Phi_t \geq (7/8) \cdot n\right] \geq \mathbf{Pr}[\Phi_t(v_t) = 1 \mid \Phi_t \geq (7/8) \cdot n] \cdot \mathbf{Pr}\left[v_t \underset{\mathcal{B}}{\rightsquigarrow} w_t \mid \Phi_t(v_t) = 1\right] \tag{8.3}$$

$$\geq \left(\frac{1}{n} \cdot \Phi_t\right) \cdot \left(\frac{1}{n} \cdot (1/8) \cdot n\right) \geq \left(\frac{1}{n} \cdot \frac{7n}{8}\right) \cdot \left(\frac{1}{n} \cdot (1/8) \cdot n\right) \tag{8.4}$$

$$= \frac{7}{64} \geq \frac{1}{16} \tag{8.5}$$

Thus, if the potential has not dropped after T steps, the expected number of crossings is $(1/16) \cdot T$. Using standard Chernoff-like tail estimates, we can show that for any $T \in \Omega(\log n)$, this would imply that at least $\frac{T}{32}$ paths crossed $\mathcal{B}$, w.h.p. Thus, within $T \in O(\epsilon^{-1} \cdot k \cdot c)$ steps, we either reached a step with low potential or had sufficiently many crossings to remove $\mathcal{B}$ with probability $1 - O(e^{-c})$. This proves Lemma 8.1.

Following Lemma 8.1, after $T \in O(\epsilon^{-1} \cdot k \cdot \log n)$ iterations, the algorithm has found the desired separator, w.h.p. It remains to prove the proclaimed runtime, i.e., that the algorithm uses $\tilde{O}(\epsilon^{-1} \cdot k \cdot \log n)$ minor aggregations and $O(\epsilon^{-1} \cdot k \cdot \log n)$ 2-approximate SetSSP computations, w.h.p. To this end, we show that each iteration of the algorithm can be implemented with $\tilde{O}(1)$ minor aggregations and two approximate SetSSP computations. Formally, we prove the following

Lemma 8.3 (Complexity). *A single iteration of the algorithm can be implemented with $\tilde{O}(1)$ minor aggregations and two 2-approximate SetSSP computations.*

Proof. All five steps of the algorithm clearly contain either $\tilde{O}(1)$ minor aggregations or a 2-approximate SETSSP computation. For the sake of completeness, we go through them one by one.

- Step 1 can be implemented by simply aggregating the maximum of a set of random values. We let each node $v \in V_{t-1}$ draw a random number $r_v \in [1, n^c]$ for a constant $c > 2$ uniformly and independently at random. All nodes draw unique numbers for a large enough c, w.h.p. Given that there are no duplicate numbers, we aggregate the maximum of all these numbers and pick the node with the highest number as $v_t := \arg\max_{v \in V} r_v$.
- Step 2 is a 2-approximate shortest path computation from the previously sampled node v_t. Note that we sample at most one node per connected component.
- Step 3 consists of another maximum aggregation and $\tilde{O}(1)$ aggregations to implement the descendant aggregation from Lemma 7.10. More precisely, we can sample a random node from $w_t N_{v_t}$ by letting each node $w \in N_{v_t}$ draw a random number uniformly and independently at random and then aggregate the maximum. Let now $P_t := (v_t, \dots, w_t)$ be the path from v_t to w_t in T_{v_t}. All nodes on this path need to learn that they are on this path. We use tree operations from Lemma 7.10 for this. Node w_t chooses 1 as its private input; all others choose 0. Now, each node computes the sum of all its descendants' inputs. Recall that by Lemma 7.10, all nodes can compute aggregate functions on all their descendants with $\tilde{O}(1)$ aggregations. This sum is 1 for all nodes on P_t and 0 for all others. Here, we exploited that v_t is the tree's root (otherwise, it would not be true, and we would require a second aggregation).
- Step 4 is another 2-approximate SETSSP computation from set P_t. Again, we sample at most one set per connected component.
- Finally, Step 5 is a purely local operation. Thus, it requires neither aggregations nor SETSSP computations.

Therefore, a step of the algorithm can be executed within $\tilde{O}(1)$ rounds of minor aggregation and exactly two 2-approximate SETSSP computations as claimed. □

Together, these two results imply Theorem 6. For $T \in O(\epsilon^{-1} \cdot k \cdot \log n)$, Lemma 8.1 implies that we sample a separator, w.h.p., and Lemma 8.3 implies that the T can be implemented efficiently in HYBRID and CONGEST.

8.3.1 PROOF OF LEMMA 8.2

In this section, we prove Lemma 8.2. We show that for a weighted k-path separable graph G of diameter $\mathcal{D}$ there exists a set of paths $\mathcal{B}$ that intersects with a constant fraction of paths of length at most $2\mathcal{D}$. We begin our proof with the following auxiliary lemma:

Lemma 8.4. *Let $G := (V, E, w)$ be a weighted k-path separable graph of n nodes with (weighted) diameter $\mathcal{D}$. Then, there exists a set $\mathcal{Z} \subseteq V$ with $|\mathcal{Z}| \geq (3/4) \cdot n$ and a set $\mathcal{B} = \{P_1, \dots, P_\kappa\}$ of at most k simple path of length $32\mathcal{D}$ such that for all $v \in \mathcal{Z}$, it holds $B_{G \setminus \mathcal{B}}(v, 8\mathcal{D}) \leq (3/4) \cdot n$.*

Proof. Define $\mathcal{D}' = 8\mathcal{D}$. To construct set $\mathcal{Z}$, we will first establish some helpful properties of k-path separable graphs. Our first important concept is the *critical index* of a k-path separator. We define the **critical index** $i_{\mathcal{D}'}$ to be the highest index, $1 \leq i_{\mathcal{D}'} \leq k'$, for which it holds:

$$\exists v \in V : |B_{G^{(i_{\mathcal{D}'})}}(v, \mathcal{D}')| > (3/4) \cdot n$$

and

$$\nexists v \in V : |B_{G^{(i_{\mathcal{D}'}+1)}}(v, \mathcal{D}')| > (3/4) \cdot n$$

One can easily verify that there is a well-defined $i_{\mathcal{D}'}$ for each $\mathcal{D}' \geq \mathcal{D}$. Note that $\mathcal{D}'$ is greater than the graph's diameter and therefore:

$$\forall v \in V : |B_G(v, \mathcal{D}')| = n$$

Recall that by definition of k-path separators by Abraham and Gavoille in [AG06], each connected component in $G \setminus S$ is of size at most $n/2$. Thus, *any* ball of *any* size around *any* node (that has not been removed as part of the separator) can contain at most $n/2$ nodes once we removed S. Thus, if there were no critical index for a distance $\mathcal{D}'$, it would hold:

$$\exists v \in V : |B_{G \setminus S}(v, \mathcal{D}')| > (3/4) \cdot n$$

This is a contradiction as all connected components in $G \setminus S$ are of size at most $n/2$. From now on, we are interested in these first $i_{\mathcal{D}'}$ (sets of) paths of the separator S, which we denote as

$$\mathcal{S}' := (\mathcal{P}_1, \ldots, \mathcal{P}_{i_{\mathcal{D}'}}).$$

Note that it holds for all $v \in V \setminus \mathcal{S}'$:

$$B_{G \setminus \mathcal{S}'}(v, \mathcal{D}') \leq (3/4) \cdot n$$

Thus, the set $\mathcal{S}'$ is a weak $\mathcal{D}'$-separator. We will refer to it as the critical separator. Further, we call the last *unseparated* subgraph $G^{(i_{\mathcal{D}'})}$ the critical graph. Next, we introduce another central element of our proof, the *terminal node*, which is a node that still has more than $(3/4) \cdot n$ nodes in the distance $\mathcal{D}'$ in $G^{(i_{\mathcal{D}'}-1)}$. As there can be more than one node with this property, we pick the node with the highest identifier. However, any other method to break ties would also work. Formally, we say a node $z_{\mathcal{D}'} \in V$ is a terminal node if and only if it holds:

$$z_{\mathcal{D}'} := \underset{z \in V}{\arg\max} \{z \in V \mid \forall j \leq i_{\mathcal{D}'} : |B_{G^{(j)}}(z, \mathcal{D}')| \geq (3/4) \cdot n\}$$

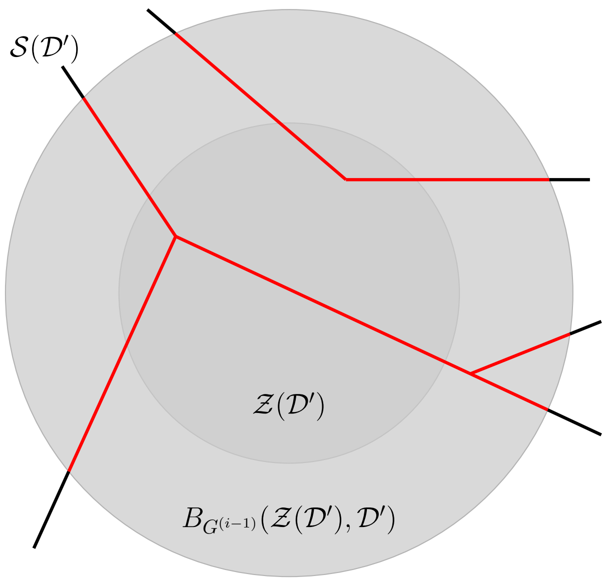

$\mathcal{S}(\mathcal{D}')$
$\mathcal{Z}(\mathcal{D}')$
$B_{G^{(i-1)}}(\mathcal{Z}(\mathcal{D}'), \mathcal{D}')$

Figure 8.2: An illustration of the boundary $\mathcal{B}(\mathcal{Z}(\mathcal{D}'))$. The inner circle denotes the core set $\mathcal{Z}(\mathcal{D}')$ and the outer circle denotes the ball $B_{G(i-1)}(\mathcal{Z}(\mathcal{D}'), \mathcal{D}')$. $\mathcal{S}(\mathcal{D}')$ is the critical separator, consisting of several shortest paths. The red parts of $\mathcal{S}(\mathcal{D}')$ make up the boundary $\mathcal{B}(\mathcal{Z}(\mathcal{D}'))$. It suffices to remove the boundary s.t. a constant fraction of nodes have a distance greater than $\mathcal{D}'$ from the core set.

Given the concept of terminal nodes and critical indices, we can now define our set $\mathcal{Z}$ as

$$\mathcal{Z} := B_{G^{(i_{\mathcal{D}'})}}(z_{\mathcal{D}'}, \mathcal{D}')$$

Clearly, $\mathcal{Z}$ contains at least $(3/4) \cdot n$ nodes by the definition of the terminal node. Thus, we already have the correct size required by the lemma. Therefore, it remains to prove the existence of the set $\mathcal{B}$. We define it as the intersections of all balls $B_{G^{(i)}}(z, \mathcal{D}')$ of all nodes $z \in \mathcal{Z}$ with all paths of $\mathcal{S}'$ from the respective graphs. Formally, the boundary is defined as follows:

Definition 8.2 (Boundary). *Let $\mathcal{S}(\mathcal{D}') := (\mathcal{P}_1, \dots, \mathcal{P}_{i_{\mathcal{D}'}})$ with $\mathcal{P}_i := \{P_{i_1}, P_{i_2}, \dots\}$ be the critical separator and let $\mathcal{Z}$ be the core set. Then, we define the boundary $\mathcal{B} := \{\mathcal{P}'_1, \mathcal{P}'_2, \dots\}$ as follows:*

$$P'_{i_j} := P_{i_j} \cap B_{G^{(i-1)}}(\mathcal{Z}, \mathcal{D}')$$
$$\mathcal{P}'_i := \{P'_{i_1}, P'_{i_2}, \dots\}$$
$$\mathcal{B} := \{\mathcal{P}'_1, \mathcal{P}'_2, \dots\}$$

Figure 8.2 depicts the intuition behind this definition. Note that the boundary is a strict subset of a k-path separator. Therefore, there can be at most k sets P'_{i_j}, one for each path P_{i_j} in the separator. However, the sets P'_{i_j} themselves are not necessarily connected paths. They are merely subsets of paths that do not necessarily consist of consecutive nodes of that path. We will show that for each P'_{i_j} there is a path of length at most $4\mathcal{D}'$ that covers all its nodes. More formally, it holds:

Claim 8. *Let $G := (V, E, w)$ be a weighted k-path separable graph let $\mathcal{B}(\mathcal{Z}(\mathcal{D}')) := \{\mathcal{P}'_1, \mathcal{P}'_2, \dots\}$ with $\mathcal{P}'_i = \{P'_{i_1}, P'_{i_2}, \dots\}$ be as defined in Definition 8.2. Then, for all P'_{i_j}, there is a simple path $\tilde{P}_{i_j}$, s.t., it holds:*

$$\ell(\tilde{P}_{i_j}) \leq 4\mathcal{D}'$$

Proof. Let P_{i_j} be a path from the critical separator $\mathcal{S}$. In the following, we denote $v_{(1)}$ as the first node of P_{i_j} that intersects with $B_{G^{(i-1)}}(\mathcal{Z}, \mathcal{D}')$. Likewise, let $v_{(\ell')}$ be the last node of P_{i_j} that intersects with $B_{G^{(i-1)}}(\mathcal{Z}, \mathcal{D}')$. Thus, all nodes of P_{i_j} that intersect with $B_{G^{(i-1)}}(\mathcal{Z}, \mathcal{D}')$ must lie on the subpath $\tilde{P}_{i_j} := (v_{(1)}, \dots, v_{(\ell')})$. As it covers all nodes, it remains bound to its length. Recall that P_{i_j} is a shortest path in $G^{(i-1)}$ and therefore, it holds that:

$$d_{P_{i_j}}(v_{(1)}, v_{(\ell')}) = d_{G^{(i-1)}}(v_{(1)}, v_{(\ell')}).$$

For both $v_{(1)}$ and $v_{(\ell')}$ there must be (not necessarily distinct) nodes $v', w' \in \mathcal{Z}$ where $v_{(1)} \in B_{G^{(i-1)}}(v', \mathcal{D}')$ and $v_{(\ell')} \in B_{G^{(i-1)}}(w', \mathcal{D}')$. By definition, both v' and w' are in distance $\mathcal{D}'$ to terminal node $z_{\mathcal{D}'}$ in the critical graph $G^{(i_{\mathcal{D}'})}$. Since we only remove nodes from G, it holds:

$$B_{G^{(1)}}(z_{\mathcal{D}'}, \mathcal{D}') \supseteq B_{G^{(2)}}(z_{\mathcal{D}'}, \mathcal{D}') \supseteq \dots \supseteq B_{G^{(i_{\mathcal{D}'})}}(z_{\mathcal{D}'}, \mathcal{D}')$$

Thus, we can upper bound the distance between v' and w' in $G^{(i-1)}$ by their distance in $G^{(i_{\mathcal{D}'})}$. By the triangle inequality, it therefore holds for all $v', w' \in \mathcal{Z}(\mathcal{D}')$:

$$d_{G^{(i-1)}}(v', w') \leq d_{G^{(i_{\mathcal{D}'}-1)}}(v', w') \leq d_{G^{(i_{\mathcal{D}'}-1)}}(v', z_{\mathcal{D}'}) + d_{G^{(i_{\mathcal{D}'}-1)}}(z_{\mathcal{D}'}, w') \leq 2\mathcal{D}'$$

Thus, both v' and w' are in the distance at most $2\mathcal{D}'$ in graph $G^{(i-1)}$ where P_{i_j} is a shortest path. Combining these facts with the triangle inequality gives us:

$$\begin{aligned} d_{G^{(i-1)}}(v_{(1)}, v_{(\ell')}) &\leq d_{G^{(i-1)}}(v_{(1)}, v') + d_{G^{(i-1)}}(v', w') + d_{G^{(i-1)}}(w', v_{(\ell')}) \\ &\leq \mathcal{D}' + d_{G^{(i-1)}}(v', w') + \mathcal{D}' \\ &\leq 2\mathcal{D}' + 2\mathcal{D}' \leq 4\mathcal{D}' \end{aligned}$$

Thus, the claim follows. □

Next, we claim that this boundary must intersect with many paths of length $\mathcal{D}'$ as removing the boundary causes all nodes in $\mathcal{Z}$ to lose the connection to a constant fraction of nodes.

Claim 9. *For all $z \in \mathcal{Z} \setminus \mathcal{B}$, it holds $B_{G \setminus \mathcal{B}}(z, \mathcal{D}') \leq (3/4) \cdot n$.*

Proof. Recall that $\mathcal{S}'$ is weak $\mathcal{D}'$-separator and therefore, it holds for all nodes $v \in V \setminus \mathcal{S}'$ by definition:

$$|B_{G \setminus \mathcal{S}'}(v, \mathcal{D}')| \leq (3/4) \cdot n$$

The boundary definition contains all nodes of $\mathcal{S}'$ that intersects with a path of length $\mathcal{D}'$ from any node in $\mathcal{Z}$. Therefore, for all nodes $z \in \mathcal{Z}$, it holds:

$$|B_{G \setminus \mathcal{S}'}(v, \mathcal{D}')| = |B_{G \setminus \mathcal{B}}(v, \mathcal{D}')|$$

This can be asserted by a simple induction over the sets $\mathcal{P}'_1, \dots, \mathcal{P}'_{i_{\mathcal{D}'}}$ and $\mathcal{P}_1, \dots, \mathcal{P}_{i_{\mathcal{D}'}}$ that make up $\mathcal{B}$ and $\mathcal{S}$ respectively. For easier notation, define:

$$G_i := G \setminus \bigcup_{j<i} \mathcal{P}_j$$
$$G'_i := G \setminus \bigcup_{j<i} \mathcal{P}'_j$$

For the induction beginning, we suppose that no paths are removed. In this case, the statement holds trivially as $G'_1 = G_1$.

For the induction steps, we assume that for all $z \in \mathcal{Z}$, it holds:

$$B_{G_i}(z, \mathcal{D}') = B_{G'_i}(z, \mathcal{D}')$$

We now show that $B_{G_{i+1}}(z, \mathcal{D}') = B_{G'_{i+1}}(z, \mathcal{D}')$ for all $z \in \mathcal{Z}$. As each P'_{i_j} is a subset of P_{i_j}, it holds that $B_{G_{i+1}}(v, \mathcal{D}') \subseteq B_{G'_{i+1}}(v, \mathcal{D}')$ as we can only reach more nodes from z we remove fewer nodes. For contradiction, assume there is a node $u \in B_{G'_{i+1}}(z, \mathcal{D}')$ that is not in $B_{G_{i+1}}(z, \mathcal{D}')$. As $u \notin B_{G_{i+1}}(z, \mathcal{D}')$, a node on **every** path from z to u of length at most $\mathcal{D}'$ is removed. In other words, every such path intersects with some path $P_{i_j} \in \mathcal{P}_i$. Let p be such an intersecting node. Clearly, $p \in B_{G_i}(z, \mathcal{D}')$ as p is closer to z than u because it is a predecessor of u on a path of length $\mathcal{D}'$. Therefore, any such p also part of $P'_{i_j} := B_{G'_i}(z, \mathcal{D}') \cap P_{i_j}$ as, per the induction's hypothesis, it holds $B_{G'_i}(z, \mathcal{D}') = B_{G_i}(z, \mathcal{D}')$. Thus, all possible nodes p are removed from G'_i and u cannot not reachable in G'_{i+1} from z within radius $\mathcal{D}'$. This is a contradiction as we assumed $u \in B_{G'_{i+1}}(z, \mathcal{D}')$. Therefore, it holds $B_{G_{i+1}}(v, \mathcal{D}') \supseteq B_{G'_{i+1}}(v, \mathcal{D}')$ and together with our initial observation that $B_{G_{i+1}}(v, \mathcal{D}') \subseteq B_{G'_{i+1}}(v, \mathcal{D}')$, we have $B_{G_{i+1}}(v, \mathcal{D}') = B_{G'_{i+1}}(v, \mathcal{D}')$ as claimed.

This concludes the proof of the claim as

$$B_{G \setminus \mathcal{S}'}(z, \mathcal{D}') = B_{G_{i_{\mathcal{D}'}}}(z, \mathcal{D}') = B_{G'_{i_{\mathcal{D}'}}}(z, \mathcal{D}') = B_{G \setminus \mathcal{B}}(z, \mathcal{D}')$$

□

Thus, the sets $\mathcal{Z}$ and $\mathcal{B}$ fulfill the properties required by the lemma. As $\mathcal{D}' = 8\mathcal{D}$, $\mathcal{B}$ consists of (at most) k paths of length $32\mathcal{D}$. Further, as for each $z \in Z$ it holds

$$B_{G \setminus \mathcal{B}}(z, 8\mathcal{D}) = B_{G \setminus \mathcal{S}'}(z, 8\mathcal{D}) \leq (3/4) \cdot n.$$

Thus, the lemma follows. □

Next, we require the following helpful observation that is true for any distance metric on graphs

Lemma 8.5. *Let $G := (V, E, w)$ be a weighted graph and let $\mathcal{D}'$ be some distance parameter. Suppose there is a set of nodes $\mathcal{Z} \subseteq V$ of size $|\mathcal{Z}| \geq (1/c) \cdot n$ for some $c > 1$, s.t., it holds for all $v \in \mathcal{Z}$ that $|B_G(v, 2\mathcal{D}')| \leq (1 - 1/c) \cdot n$. Then,* ***for all nodes*** *$v \in V$, it holds:*

$$|B_G(v, \mathcal{D}')| \leq (1 - 1/c) \cdot n$$

Proof. The lemma clearly holds for all nodes in $\mathcal{Z}$, so it suffices to analyze the remaining nodes. Assume for contradiction that there is a node $w \in V \setminus \mathcal{Z}$ with $|B(w, \mathcal{D}')| > (1 - 1/c) \cdot n$. Then, $B(w, \mathcal{D}')$ must contain at least one node $v \in \mathcal{Z}$, which — by the very definition of $B(w, \mathcal{D}')$ — implies $d(v, w) \leq \mathcal{D}'$. By the triangle inequality, it then holds for every other node $u \in B(w, \mathcal{D}')$ that

$$d(v, u) \leq d(v, w) + d(w, u) \leq 2\mathcal{D}'. \tag{8.6}$$

Therefore, it follows $B(w, \mathcal{D}') \subseteq B(v, 2\mathcal{D}')$ As we have a clear bound on the size of $B(v, 2\mathcal{D}')$, this implies that:

$$(1 - 1/c) \cdot n < |B(w, \mathcal{D}')| \leq |B(v, 2\mathcal{D}')| \leq (1 - 1/c) \cdot n \tag{8.7}$$

This is a contradiction. □

Let sets $\mathcal{Z} \subseteq V$ and $\mathcal{B} := \{P_1, P_2, \ldots\}$ be as defined in Lemma 8.4. Note that for all $z \in \mathcal{Z}$, it holds:

$$B_{G \setminus \mathcal{B}}(z, 8\mathcal{D}) \leq (3/4) \cdot n \tag{8.8}$$

Lemma 8.5 now implies that **for all** $v \in V$ it holds $B_{G \setminus \mathcal{B}}(v, 4\mathcal{D}) \leq (3/4) \cdot n$. Therefore, for each node $v \in V$ there must be $(1/4) \cdot n$ nodes $w \in V$ such that all paths of length at most $4\mathcal{D}$ from v to w cross $\mathcal{B}$.

8.3.2 Proof of Lemma 8.1

In this section, we prove Lemma 8.1 using Lemma 8.2 and some fairly standard tools from the analysis of potential functions. Recall that we consider the following potential:

$$\Phi_t := \left|\left\{v \in V_t \;\middle|\; |B_{G_t}(v, 2\mathcal{D})| \geq (7/8) \cdot n\right\}\right|$$

We want to show that after $T \in O(\epsilon^{-1} \cdot k \cdot c)$ steps, the potential has dropped below $(7/8) \cdot n$. As soon as this potential drops below $(7/8) \cdot n$, we have more than $(1/8) \cdot n$ nodes with less than $(7/8) \cdot n$ nodes in distance $2\mathcal{D}$. Therefore, by Lemma 8.5, each node has at most $(1/8) \cdot n$ nodes in distance $\mathcal{D}$ as desired.

By Lemma 8.2, there exists a set $\mathcal{B} = \{P_1, \dots, P_\kappa\}$ with $\kappa \leq k$ simple paths of length $32\mathcal{D}$ such that for all $v \in V$, it holds $B_{G\setminus\mathcal{B}}(v, 4\mathcal{D}) \leq (3/4) \cdot n$. Thus, for each node $v \in V$ there is set $\mathcal{C}(v) \subseteq B(v, 4\mathcal{D})$, s.t., all path of length $4\mathcal{D}$ from v to $\mathcal{C}(v)$ contain a node (of a path in) $\mathcal{B}$. We denote these nodes as the cutoff nodes of v. It holds $\mathcal{C}(v) := V \setminus B_{G\setminus\mathcal{B}}(v, 4\mathcal{D})$. We want to show that — as long as the potential is high — there is a good chance to sample a path between a node v and one of its cutoff nodes. To this end, note that every node $v \in V$ with $\Phi_t(v) = 1$ that is considered in the potential *must* have $(1/8) \cdot n$ cutoff nodes in distance $2\mathcal{D}$. This follows from the definition of the potential: Recall that each node has $(1/4) \cdot n$ cutoff nodes. Further, there are at most $(1/8) \cdot n$ nodes in distance greater than $2\mathcal{D}$. Otherwise, it would hold $\Phi_t(v) = 0$. Even if all these nodes are cutoff nodes, there must still be $(1/4) \cdot n - (1/8) \cdot n = (1/8) \cdot n$ cutoff nodes in the distance at most $2\mathcal{D}$. Further, As we compute 2-approximate shortest paths to all nodes, the approximate distance to these nodes is at most $4 \cdot \mathcal{D}$. Therefore, all cutoff nodes of v are the potential endpoints of the approximate shortest path we sample. We may not choose the shortest path (of length less than $2\mathcal{D}$), but the length of the path we pick is at most $4\mathcal{D}$ and (by definition) crosses $\mathcal{B}$.

Thus, as each node has $n/8$ cutoff nodes in distance $2\mathcal{D}$, there is chance of $1/8$ to sample a cutoff node. This means, as long as $\Phi_t \geq p \cdot n$, the probability that we sample a path that crosses $\mathcal{B}$ is at least $p/8$. Intuitively, this means that after carving T paths, **either** the potential has dropped, **or** there are $\Theta(T)$ paths that intersect with $\mathcal{B}$ on expectation. Note that every time the latter happens, we remove a chunk of length $(\epsilon/2)\mathcal{D}$ from some path in $\mathcal{B}$. However, this cannot happen often due to these paths' bounded length of $32\mathcal{D}$; therefore, the potential must drop quickly.

We now want to formalize this intuition. To this end, fix a path $P := (v_1, v_2, \dots)$ in graph G and denote $\mathcal{K}_t(P)$ as the number of sets from $K_1, \dots, K_t$ that intersect with P. Formally, it holds:

$$\mathcal{K}_t(P) := \sum_{i \leq t} \mathbb{1}_{K_i \cap P \neq \emptyset}$$

Now define the number of clusters that intersect with $\mathcal{B}$ as

$$\mathcal{K}_t := \sum_{P_i \in \mathcal{B}} \mathcal{K}_t(P_i)$$

More formally, it holds:

Lemma 8.6. *After carving $\tau \geq c \cdot 768$ sets $K_1, \dots, K_\tau$, it holds:*

$$\mathbf{Pr}[\{\Phi_\tau \geq (7/8) \cdot n\} \cap \{\mathcal{K}_\tau \leq \tau/256\}] \leq e^{-c}$$

Proof. First, we denote $\delta\mathcal{K}_t := \mathcal{K}_t - \mathcal{K}_{t-1}$ as the difference in the number of intersections from step $t-1$ to step t. Note that this difference is always non-negative as the number of balls intersecting with the separator can not decrease. Equipped with this definition, we want to count the number of rounds in which we either increase $\mathcal{K}_t$, or Φ_t falls below $(7/8) \cdot n$. We refer to these rounds as *good* steps. Clearly, after at most $\rho\tau$ goods steps (with $\rho \leq 1$) either $\rho\tau$ balls intersect with the boundary, or it holds $\Phi_t < (7/8) \cdot n$. To bound the number of good steps, we define the following indicator variable for good steps:

$$Y_t := \mathbb{1}_{\{\Phi_t < (7/8) \cdot n\} \vee \{\delta\mathcal{K}_t \geq 1\}}$$

Further, let $\mathcal{Y}_\tau := \sum_{i=1}^{\tau} Y_i$ the total number of good steps until some step $\tau > 0$. Next, we will show that the probability that a given step is good is constant regardless of the processes' history. That is, regardless of what happened in the first $t-1$ steps, the probabalility that step t is good is at least $1/16$. First, we note that the probability of whether a step t is good is completely determined by graph G_{t-1}. To this end, let $\mathcal{G}_{t-1}$ be a random variable that denotes the active subgraph in step t. Let $G_{t-1} \subset G$ the graph at the beginning of step t (which is created by all choices made by the algorithm in steps $1, \dots, t-1$). We will now bound the probability that we sample a path that crosses $\mathcal{B}$.

Claim 10. *For all graphs $G_t \subseteq G$, it holds:*

$$\mathbf{Pr}[Y_t \mid \mathcal{G}_{t-1} = G_t] \geq \frac{\Phi(G_t)}{8n} \tag{8.9}$$

Proof. During this proof, we will write $\mathbf{Pr}[X \mid G_t]$ instead of $\mathbf{Pr}[X \mid \mathcal{G}_{t-1} = G_t]$ for brevity. Further, let $P_t := (v_t, \dots, w_t)$ be the path sampled in step t and let v_t and w_t be its respective start and endpoint. As the algorithm only removes nodes and edges, the path P_t must also exist in the original graph G. Therefore, the path must cross $\mathcal{B}$ if v_t is a critical node and w_t is one of its cutoff nodes. In the following fix a graph $\mathcal{G}_{t-1} = G_{t-1}$. Recall that the cutoff nodes $\mathcal{C}(v) := V \setminus B_{G \setminus \mathcal{B}}(v, 4\mathcal{D})$ are all nodes, s.t., all paths (of length at most $4 \cdot \mathcal{D}$) between v and $w \in \mathcal{C}(v)$ contain a node of $\mathcal{B}$. Hence, if we sample a cutoff node, $\mathcal{K}_t$ increases by one. Denote this event as $\{v_t \underset{\mathcal{B}}{\rightsquigarrow} w_t\}$. Further, denote the critical nodes conditioned on G_{t-1} as

$$C(G_{t-1}) = \left\{ v \in V_t \;\middle|\; |\mathcal{C}(v) \cap B_{G_{t-1}}(v, 2\mathcal{D})| \geq n/8 \right\} \tag{8.10}$$

Note that $C(G_{t-1})$ is completely determined by G_{t-1} and therefore is not a random variable. Given this definition, note that it holds:

$$\mathbb{E}\left[\delta\mathcal{K}_t \geq 1 \mid G_{t-1}\right] = \mathbb{E}\left[v_t \underset{\mathcal{B}}{\rightsquigarrow} w_t \mid G_{t-1}\right] \tag{8.11}$$

$$\geq \sum_{v \in V} \mathbb{1}_{v \in C(G_{t-1})} \cdot \mathbf{Pr}[v_t = v \mid G_{t-1}] \cdot \mathbf{Pr}[w_t \in \mathcal{C}(v) \mid v_t = v \cap G_{t-1}] \tag{8.12}$$

$$\geq \frac{1}{n} \cdot \sum_{v \in V} \mathbb{1}_{v \in C(G_{t-1})} \cdot \mathbf{Pr}[w_t \in \mathcal{C}(v) \mid v_t = v \cap G_{t-1}] \tag{8.13}$$

Recall that by definition, each critical node has at least $n/8$ cutoff nodes in distance $2\mathcal{D}$. By the properties of the approximate shortest path algorithm, the approximate distance to all these nodes is $4\mathcal{D}$ and, thus, all these nodes are considered in the sampling. Further, as we use approximate shortest paths that can only overestimate the actual distance, we sample from a subset of all nodes in the distance $4\mathcal{D}$. Note that there can obviously be at most n nodes in distance $4\mathcal{D}$. Therefore, the probability that one v_t's cutoff nodes are picked as the path's endpoint w_t is:

$$\mathbf{Pr}[w_t \in \mathcal{C}(v) \mid v_t = v \cap G_{t-1}] := \frac{\left|\mathcal{C}(v) \cap B_{G_{t-1}}(v, 2\mathcal{D})\right|}{\left|B_{G_{t-1}}(v, 4\mathcal{D})\right|} \geq \frac{n}{8n} = \frac{1}{8} \tag{8.14}$$

Putting this back in the previous formula, we get:

$$\mathbb{E}\left[v_t \underset{\mathcal{B}}{\rightsquigarrow} w_t \mid G_{t-1}\right] \geq \frac{1}{n} \cdot \sum_{v \in V} \mathbf{Pr}[v \in C_t \mid G_{t-1}] \cdot \mathbf{Pr}[w_t \in \mathcal{C}(v) \mid v_t = v] \tag{8.15}$$

$$\geq \frac{1}{n} \cdot \sum_{v \in V} \mathbf{Pr}[v \in C_t \mid G_{t-1}] \cdot \frac{1}{8} = \frac{|C(G_t)|}{8n} \tag{8.16}$$

Finally, observe that for each node $v \in V$ with $|B_{G_{t-1}}(v, 2\mathcal{D})| \geq (7/8) \cdot n$, it also holds that $|\mathcal{C}(v) \cap B_{G_{t-1}}(v, 2\mathcal{D})| \geq n/8$. Thus, we have $\Phi'_{t-1} \leq C(G_{t-1})$. This proves the claim □

Let now $\mathcal{H}_{t-1}$ be an *arbitrary* set of events that happened in the first $t-1$ steps of the algorithm. In particular, all of these events may only affect the topology $\mathcal{G}_t$ in step t but not the actual choice of the path in step t. Given this definition, we show the following:

Claim 11. $\mathbb{E}\left[Y_t \mid \mathcal{H}_{t-1}\right] \geq 7/64$

Proof. To simplify notation, define Φ'_{t-1} to be the potential at the beginning of step t given $\mathcal{H}_{t-1}$, i.e., it holds:

$$\{\Phi'_{t-1} = \eta\} := \{\Phi_{t-1} = x\} \cap \mathcal{H}_{t-1} \tag{8.17}$$

Analogously, Φ'_t will be the number of critical nodes in step t conditioned on H_{t-1}. Note that both Φ'_t and Φ'_{t-1} is random variable as $\mathcal{H}_{t-1}$ may not fully determines G_{t-1} and thus the corresponding potential. By the law of total expectation, it holds:

$$\mathbb{E}\left[Y_t \mid \mathcal{H}_{t-1}\right] = \mathbf{Pr}\left[\Phi'_{t-1} < \frac{7n}{8}\right] \cdot \mathbb{E}\left[Y_t \mid \Phi'_{t-1} < \frac{7n}{8}\right] + \mathbf{Pr}\left[\Phi'_{t-1} \geq \frac{7n}{8}\right] \cdot \mathbb{E}\left[Y_t \mid \Phi'_{t-1} \geq \frac{7n}{8}\right] \tag{8.18}$$

Recall that the potential is monotonially decreasing. Thus, of Φ_{t-1} is smaller than $(7/8) \cdot n$, then Φ_t must also be smaller than $(7/8) \cdot n$. Therefore, we have $\mathbb{E}\left[Y_t \mid \Phi'_{t-1} < \frac{7n}{8}\right] = 1$. As Y_t is a binary random variable that can only take values 0 and 1, it furthermore holds:

$$\begin{aligned}
\mathbb{E}\left[Y_t \mid \Phi'_{t-1} < \frac{7n}{8}\right] &= 1 \\
&\geq 1 \cdot \mathbf{Pr}\left[Y_t = 1 \mid \Phi'_{t-1} \geq \frac{7n}{8}\right] + 0 \cdot \mathbf{Pr}\left[Y_t = 0 \mid \Phi'_{t-1} \geq \frac{7n}{8}\right] \\
&:= \mathbb{E}\left[Y_t \mid \Phi'_{t-1} \geq \frac{7n}{8}\right]
\end{aligned}$$

Therefore, it holds:

$$\begin{aligned}
\mathbb{E}\left[Y_t \mid G_{t-1}\right] &= \mathbf{Pr}\left[\Phi'_{t-1} < \frac{7n}{8}\right] \cdot 1 + \mathbf{Pr}\left[\Phi'_{t-1} \geq \frac{7n}{8}\right] \mathbb{E}\left[Y_t \mid \Phi'_t \geq \frac{7n}{8}\right] && (8.19) \\
&\geq \mathbf{Pr}\left[\Phi'_{t-1} < \frac{7n}{8}\right] \mathbb{E}\left[Y_t \mid \Phi_t \geq \frac{7n}{8}\right] + \mathbf{Pr}\left[\Phi'_{t-1} \geq \frac{7n}{8}\right] \mathbb{E}\left[Y_t \mid \Phi_t \geq \frac{7n}{8}\right] && (8.20) \\
&\geq \left(\mathbf{Pr}\left[\Phi'_{t-1} < \frac{7n}{8}\right] + \mathbf{Pr}\left[\Phi'_{t-1} \geq \frac{7n}{8}\right]\right) \cdot \mathbb{E}\left[Y_t \mid \Phi_t \geq \frac{7n}{8}\right] && (8.21) \\
&= \mathbb{E}\left[Y_t \mid \Phi'_{t-1} \geq \frac{7n}{8}\right] \geq \mathbb{E}\left[\delta\mathcal{K}_t \geq 1 \mid \Phi'_{t-1} \geq \frac{7n}{8}\right] && (8.22)
\end{aligned}$$

Thus, to the claim, it suffices to consider the case where $\Phi'_{t-1} \geq (7/8) \cdot n$ and to bound the probability that we sample a path that crosses $\mathcal{B}$.

For a fixed value $\eta \in [0, n]$ of the potential, it holds by the law of total expectation:

$$\mathbb{E}\left[\delta\mathcal{K}_t \geq 1 \mid \Phi'_{t-1} = \eta\right] := \sum_{\substack{G_t \subseteq G \\ \Phi(G_t)=\eta}} \mathbf{Pr}\left[G_{t-1} \mid \Phi'_{t-1} = \eta\right] \cdot \mathbb{E}\left[\delta\mathcal{K}_t \geq 1 \mid \{\mathcal{G}_{t-1} = G_{t-1}\} \cap \mathcal{H}_{t-1}\right] \tag{8.23}$$

$$:= \sum_{\substack{G_t \subseteq G \\ \Phi(G_t)=\eta}} \mathbf{Pr}\left[G_{t-1} \mid \Phi'_{t-1} = \eta\right] \cdot \mathbb{E}\left[\delta\mathcal{K}_t \geq 1 \mid \{\mathcal{G}_{t-1} = G_{t-1}\}\right] \tag{8.24}$$

$$\text{By Claim } 10: \tag{8.25}$$

$$\geq \sum_{\substack{G_t \subseteq G \\ \Phi(G_t)=\eta}} \mathbf{Pr}\left[G_{t-1} \mid \Phi'_{t-1} = \eta\right] \cdot \frac{\eta}{64} \tag{8.26}$$

$$= \frac{\eta}{8n} \cdot \sum_{\substack{G_t \subseteq G \\ \Phi(G_t)=\eta}} \mathbf{Pr}\left[G_{t-1} \mid \Phi'_{t-1} = \eta\right] \tag{8.27}$$

$$= \frac{\eta}{8n} \tag{8.28}$$

Here, the second equality follows from the fact that $\mathcal{H}_{t-1}$ does not affect the choice that the algorithm makes in step t. The last equality follows as $\sum_{\substack{G_t \subseteq G \\ \Phi(G_t)=\eta}} \mathbf{Pr}\left[G_{t-1} \mid \Phi'_{t-1} = \eta\right] = 1$.

By applying the law of total expectations again, we obtain the following lower lower bound given that the potential is greater than $(7/8) \cdot n$:

$$\mathbb{E}\left[\delta\mathcal{K}_t \mid \Phi'_{t-1} \geq \frac{7n}{8}\right] \geq \sum_{\eta=(7/8)\cdot n}^{n} \mathbf{Pr}\left[\Phi_{t-1} = \eta \mid \Phi'_{t-1} \geq \frac{7n}{8}\right] \cdot \mathbb{E}\left[v_t \underset{\mathcal{B}}{\rightsquigarrow} w_t \mid \Phi'_{t-1} = \eta\right] \tag{8.29}$$

$$\text{By Ineq. } (8.28): \tag{8.30}$$

$$\geq \sum_{\eta=(7/8)\cdot n}^{n} \mathbf{Pr}\left[\Phi_{t-1} = \eta \mid \Phi'_{t-1} \geq \frac{7n}{8}\right] \cdot \frac{\eta}{8n} \tag{8.31}$$

$$\text{Using } \eta \geq \frac{7n}{8}: \tag{8.32}$$

$$\geq \sum_{\eta=(7/8)\cdot n}^{n} \mathbf{Pr}\left[\Phi_{t-1} = \eta \mid \Phi'_{t-1} \geq \frac{7n}{8}\right] \cdot \frac{7n}{8 \cdot 8 \cdot n} \tag{8.33}$$

$$\geq \frac{7}{64} \cdot \sum_{\eta=(7/8)\cdot n}^{n} \mathbf{Pr}\left[\Phi_{t-1} = \eta \mid \Phi'_{t-1} \geq \frac{7n}{8}\right] \tag{8.34}$$

$$= \frac{7}{64} \tag{8.35}$$

The last inequality follows from the fact that $\sum_{\eta=(7/8)\cdot n}^{n} \mathbf{Pr}\left[\Phi_{t-1} = \eta \mid \Phi'_{t-1} \geq \frac{7n}{8}\right] = 1$. Thus, by Inequality (8.22), it holds:

$$\mathbb{E}\left[Y_t \mid G_{t-1}\right] \geq \mathbb{E}\left[\delta\mathcal{K}_t \geq 1 \mid \Phi'_{t-1} \geq \frac{7n}{8}\right] \geq \frac{1}{64}$$

This proves the claim. $\square$

Thus, we have a probability bound holds regardless of all events before step t. This tells us that no matter what happens, there is always a constant probability for a good step. In particular, eventough the steps of the algorithm are not independent as events in earlier steps affect the possibilities in later steps, we can still give a good tail estimate. We need the following slightly generalized version of the Chernoff bound to finalize the proof.

Lemma 8.7 (Generalized Chernoff Bound, cf. Theorem 3.52 in [Schoo]). *Let $X_1, \dots, X_n \in \{0,1\}$ be a series of (not necessarily independent) binary random variables and define $X := \sum_{i=1}^{n} X_i$. Suppose, there is a value $\rho \in [0,1]$, s.t., for all subsets $X_{i_1}, \dots, X_{i_j}$ with $1 \leq j \leq n$, it holds:*

$$\mathbb{E}\left[\prod_{k=i_1,\dots,i_j} X_k\right] \geq \rho^j.$$

Then, for any $\delta \leq 1$, it holds:

$$\mathbf{Pr}[X < (1-\delta)\rho n] \leq e^{-\frac{\delta\rho n}{3}}$$

Let now $i_1, \dots, i_j$ be some subset of the τ steps that we make. Note that by the fact that all Y_t are binary random variables and the chain rule of conditional expectations, it holds:

$$\begin{aligned}\mathbb{E}\left[\prod_{k=i_1}^{i_j} Y_k\right] &:= \mathbf{Pr}\left[\bigcap_{k=i_1}^{i_j} Y_k = 1\right] = \prod_{k=i_1}^{i_j} \mathbf{Pr}\left[Y_k = 1 \mid \bigcap_{k'=i_1}^{k-1} Y_{k'} = 1\right] \\ &= \prod_{k=i_1}^{i_j} \mathbb{E}\left[Y_k \mid \bigcap_{k'=i_1}^{k-1} Y_{k'} = 1\right] \geq \prod_{k=i_1}^{i_j} \frac{7}{64} = \left(\frac{1}{16}\right)^j\end{aligned}$$

Here, we exploited that $\left\{\cap_{k'=i_1}^{k-1} Y_{k'} = 1\right\}$ only depends on events prior to step k and used the bound of Claim 11. Thus, the conditions of Lemma 8.7 apply to $\mathcal{Y}_\tau$ by choosing $\rho = 1/16$. Therefore, it holds for all $\tau \geq 1$ and $\delta \leq 1$ that:

$$\mathbf{Pr}[\mathcal{Y}_\tau \leq (1-\delta)\rho\tau] \leq e^{-\frac{\delta\rho\tau}{3}}$$

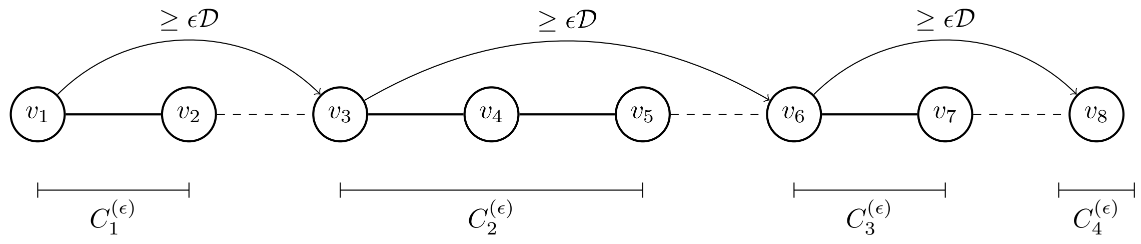


Figure 8.3: An example for the chunks $C^{(\epsilon)}(P)$ of distance ϵ of a path $P := v_1, \dots, v_8$. The solid lines denote edges between two nodes of the same chunk, the dashed lines denote edges between chunks.

For $\tau \geq c \cdot 768$, we get the following bound:

$$\begin{aligned}\mathbf{Pr}\left[\mathcal{Y}_\tau \leq \frac{\tau}{256}\right] = \mathbf{Pr}\left[\mathcal{Y}_\tau \leq \left(1-\frac{1}{2}\right)\frac{\tau}{16}\right] = \mathbf{Pr}\left[\mathcal{Y}_\tau \leq \left(1-\frac{1}{2}\right)\rho\tau\right] \\ \leq e^{-\frac{(1/2)\rho\tau}{3}} = e^{-\frac{\rho\tau}{6}} = e^{-\frac{\tau}{768}} \leq e^{-c}\end{aligned}$$

This proves the lemma. □

This verifies that if we sample many paths that cross $\mathcal{B}$ unless the potential drops down by a constant factor. However, to prove the lemma, we need more than that. Further, we need an upper bound on the number of intersections with $\mathcal{B}$. With each crossing, we remove one node from $\mathcal{B}$, so a trivial upper bound in n. However, to prove the lemma, we require a tighter bound. To this end, we show the following:

Lemma 8.8. *Consider any path P of length $32\mathcal{D}$ in G. Then for any $t \geq 0$, it holds $\mathcal{K}_t(P) \leq O(\epsilon^{-1})$.*

Proof. To this end, we introduce the notion of *chunks*. A chunk is a non-empty subpath of the path, s.t., the distance between the first node of two consecutive chunks is at least $\epsilon\mathcal{D}$. Formally, we can recursively define them as follows.

Definition 8.3 (Chunk). *Let $P := (v_1, \dots, v_m)$ be a path in graph G and let $\epsilon > 0$ be a distance parameter. Then, the chunks $C^{(\epsilon)}(P) := \left(C_1^{(\epsilon)}, \dots, C_{m'}^{(\epsilon)}\right)$ of path P are partition of P into connected subpaths. The chunks are recursively defined as follows:*

1. *The first chunk $C_1^{(\epsilon)}$ begins with node v_1.*
2. *The first node of chunk C_{i+1} is the first node in distance (greater than) $\epsilon\mathcal{D}$ to the first node of chunk C_i.*
3. *A chunk C_i contains all nodes between its first node and the first node of C_{i+1}. The last chunk contains its first node until the end of the path.*

Further, note that every time a random path P_t crosses P, we remove all nodes in the chunk it crosses. In particular, it does not matter if some nodes *between* two nodes of a chunk have already been removed as we always consider the distance with regard to G. Thus, the number of chunks is an upper bound on the number of clusters that intersect with the separator in **any** weak ball carving process.

Clearly, a path P of length $32\mathcal{D}$ consists of $128\epsilon^{-1}$ chunks of length ${}^{\epsilon}/_{4}\mathcal{D}$. This can verified as follows: Denote by $C(P)$ the chunks of length ${}^{\epsilon}/_{4}\mathcal{D}$ in P. For the sake of creating a contradiction, assume that there are more than ${}^{128}/_{\epsilon}$ chunks. Denote this number as

$$c := |C(P)|$$

The chunks $C(P)$ can be ordered $C_{(1)}, \dots, C_{(c)}$ based on their position in path P, s.t., chunks with higher order are closer to the endpoint of the path. We denote the first node of chunk $C_{(i)}$ that is part of the boundary as $v_{(i)}$. Recall that the distance between the first nodes of two neighboring chunks is at least ${}^{\epsilon}/_{4}\mathcal{D}$. If the chunks are not neighboring, the distance can only be greater. Thus, it holds:

$$d_P(v_{(i)}, v_{(i+1)}) \geq {}^{\epsilon}/_{4}\mathcal{D}$$

Let now v be the last node in the first chunk $C_{(1)}$ and $w := v_{(c)}$ be the first node in the last chunk $C_{(c)}$ that intersect with the boundary. By the definition of chunks, the distance between v and w on path P_{i_j} is at least:

$$\begin{aligned} d_P(v,w) = d_{P_{i_j}}(v, v_{(2)}) + \sum_{i=2}^{c} d_{P_{i_j}}(v_{(i)}, v_{(i+1)}) &\geq d_P(v, v_{(2)}) + \sum_{i=2}^{c} {}^{\epsilon}/_{4}\mathcal{D} = d_P(v, v_{(2)}) + (c-1){}^{\epsilon}/_{4}\mathcal{D} \\ &\geq d_P(v, v_{(2)}) + \left({}^{128}/_{\epsilon} + 1 - 1\right)\epsilon\mathcal{D} = d_{P_i}(v, v_{(2)}) + 32\mathcal{D} > 32\mathcal{D} \end{aligned}$$

This is a contradiction as P is of length $32\mathcal{D}$. □

Finally, suppose for contradiction that after $O(\epsilon^{-1} \cdot k \cdot \log n)$ iterations, the potential is still above $({}^{7}/_{8}) \cdot n$. Then, $\mathcal{K}_\mathcal{T} \in \Omega(\epsilon^{-1} \cdot k \cdot \log n)$ paths must have crossed $\mathcal{B}$, w.h.p. However, by Lemma 8.8, $\mathcal{K}_\mathcal{T}$ is bounded by $O(\epsilon^{-1} \cdot k)$. A contradiction. Thus, after $O(\epsilon^{-1} \cdot k \cdot \log n)$ iterations, the potential must be low, w.h.p., which proves the lemma.

9 Fast Construction of Separators for Planar Graphs

In this chapter, we show how to efficiently construct a path separator in (subgraphs of) a planar graph in CONGEST and the HYBRID model. In contrast to our previous weak separator construction, this is a *classical* vertex separator whose removal leaves connected components of a constant fraction of the original size. This (potentially) gives it more applications than just clustering and routing and may be seen as a tiny step towards a general framework for distributed divide-and-conquer algorithm on planar graphs. The algorithms of this chapter were first presented in the following publication:

> Jinfeng Dou, Thorsten Götte, Henning Hillebrandt, Christian Scheideler, and Julian Werthmann. Brief announcement: Distributed construction of near-optimal compact routing schemes for planar graphs. In Rotem Oshman, Alexandre Nolin, Magnús M. Halldórsson, and Alkida Balliu, editors, *Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing, PODC 2023, Orlando, FL, USA, June 19-23, 2023*, pages 67–70. ACM, 2023

Notably, for the construction of these separators, we will *not* (only) use the minor aggregation framework presented in Section 7.1.1 and instead present bespoke algorithms for CONGEST and HYBRID that use additional techniques. To be precise, our algorithm rely on the computation of so-called (planar) embeddings, a tool that is simply not (yet) available for other graph classes in the distributed setting. However, as we see, our algorithms can be easily incorporated into algorithms that are based on minor aggregations. This is because we specifically design our algorithms to also compute separators for *arbitrary* connected subgraphs of a given planar graph G. That being said, we show the following technical result in this chapter:

Theorem 7. *Path Separators for Planar Graphs*

Consider a weighted planar graph $G := (V, E, w)$ and a collection of N node-disjoint subgraphs $\mathcal{C}_1, \dots, \mathcal{C}_N \subset G$ with $C_i = (V_i, E_i)$. Then, there algorithm that constructs a 4-path separators $S_1, \dots, S_N$ of $4\ (1+\epsilon)$ approximate shortest paths in all $\mathcal{C}_1, \dots, \mathcal{C}_N$. Each connected component in $C_i \setminus S_i$ consists of at most $(3/4) \cdot |V_i|$ nodes.

The algorithm can, w.h.p., be implemented in $\tilde{O}(\epsilon^{-2} \cdot \mathrm{HD})$ time in CONGEST and $\tilde{O}(\epsilon^{-2})$ time in HYBRID.

In fact we present two different algorithms, one for each model. For both the CONGEST and the HYBRID algorithm, we base our separators on so-called *cycle separators* that can be constructed with respect to an arbitrary spanning tree T of G. These separators consist of a path in the tree T and an additional edge that connects the endpoint of the path. Constructing these separators has been well researched in the sequential, CONGEST, and PRAM model. For our construction, we exploit two useful facts about the algorithms that construct these separators. First, suppose that the tree T is rooted in some node $s \in V$. Then, for any separator path in T that goes from v to w, we can instead add the two paths from s to v and to w to the separator. This is cleary a superset of the separator and therefore a separator itself. Therefore, we can use the following non-standard definition of a cycle separator that will capture the structure of the separators we construct slightly better:

Definition 9.1 (Cycle Separator). *Let $G = (V, E)$ be a planar graph with n nodes, and let $T = (V, E_T)$ be an arbitrary spanning tree of G rooted in $s \in V$. Then, a so-called cycle separator S (with respect to T) consists of two paths $P_1 = (s, \dots, v)$ and $P_2 = (s, \dots, w)$ in T and the edge $\{v, w\} \in V^2$ that may not be part of G. All connected components in $G \setminus S$ have at most $(3/4) \cdot n$ nodes.*

Second, the algorithms that compute the separator are oblivious of the used tree, i.e., they work for arbitrary spanning tree. Thus, for our purposes, we can use a $(1+\epsilon)$-approximate spanning tree rooted in some $s \in V$. Any path in this tree is a $(1+\epsilon)$-approximate shortest path. Thus, using such a tree as a basis, we construct a separator that consists of two $(1+\epsilon)$-approximate shortest path and an edge that may or may not be in G. Since we can consider the two endpoints of an edge to be shortest path to themselves, we found a separator that consists of four approximate shortest paths.

As our main technical lemma, we show that given a suitable pre-computed spanning tree, we can, in fact, compute all separators in parallel. More precisely, we will show the following:

Lemma 9.1. *Let $G = (V, E, w)$ be a (weighted) undirected planar graph and let $\mathcal{C}_1, \dots, \mathcal{C}_N$ be a set of disjoint subgraphs of G. Further, let $\mathcal{T}$ be a spanning tree such that its subforests $T_1, \dots, T_N$ restricted to $\mathcal{C}_1, \dots, \mathcal{C}_N$ are spanning trees for $\mathcal{C}_1, \dots, \mathcal{C}_N$. In particular, each separator consists of two paths in T_i and the two endpoints of an edge possibly not contained in $\mathcal{C}_i$. Then, there is an algorithm that computes a cycle separator for each $(\mathcal{C}_i, T_i)$ simultaneously in $\tilde{O}(HD)$ time, w.h.p. in CONGEST and $\tilde{O}(1)$ time in HYBRID.*

Before we prove this lemma, let us first see how it implies Theorem 7. Note that the lemma is oblivious of the tree $\mathcal{T}$ as long as it fulfills the stated property. Thus, we may use an $(1+\epsilon)$-approximate SetSSP tree computed by the algorithm of Rozhon ⓡ al. [RGH$^+$22] with $\epsilon < 1$. Such a tree can be computed in $\tilde{O}(\epsilon^{-2} \cdot \mathrm{HD})$ time in all subgraphs simultaneously in CONGEST and $\tilde{O}(\epsilon^{-2})$ time in HYBRID. To do this, we assign *infinite* weight to all edges between components. More precisely, we choose a weight of $2Wn$ where W is the highest weight in the graph. This weight can be determined by simply aggregating the maximum of all edge weights in G. As G is planar, this can be done, w.h.p., in $\tilde{O}(\mathrm{HD})$ time in CONGEST and $\tilde{O}(1)$ time in HYBRID. After that, we determine the nodes $v_1, \dots, v_N$ of the lowest identifier in each component $\mathcal{C}_1, \dots, \mathcal{C}_N$. We call these nodes *leaders* of their respective components. Again, this can be done with a simple aggregation in each component. Recall that the subgraph is node-disjoint, and G is planar and thus this aggregation can be performed in $\tilde{O}(\mathrm{HD})$ time in CONGEST and in $\tilde{O}(\mathrm{HD})$ time in HYBRID. Finally, perform a SetSSP from all leaders using the algorithm of Rozhon ⓡ al. [RGH$^+$22] with $\epsilon < 1$. In the resulting SetSSP tree $\mathcal{T}$, each node $v \in \mathcal{C}_i$ will always be placed in the subtree of its component's leader. Since the paths to all leaders of other components must contain an edge of length $2Wn$, the leader of the $v's$ component must always be the closest if the path is at least 2-approximate. Thus, the subtrees $T_1, \dots, T_N$ of the respective leaders are spanning trees of their components $\mathcal{C}_1, \dots, \mathcal{C}_N$. Therefore, we can use these trees as input to Lemma 9.1 and obtain that consists of 4 approximate shortest paths. By definition, the two paths in T_i are approximate shortest paths, and the two endpoints of the edges are shortest paths to themselves. This proves Theorem 7.

In the remainder of this section, we, therefore, only prove Lemma 9.1. Before we start, we present some useful facts on planar graphs in Section 9.1. We split the analysis of this lemma into two parts. First, in the *hard* part, we show that the statement is true for CONGEST. This requires us to compile ideas from several papers. The main idea is to use the technique from [GP17], which only works for biconnected graphs. The algorithm was later generalized for 1-connected graphs while maintaining the $\tilde{O}(\mathrm{HD})$ runtime by [LP19]. They achieve this result by adding virtual edges to the input graph, making the augmented graph biconnected. The virtual edges can be efficiently simulated in CONGEST; therefore, simulating the original algorithm for biconnected graphs yields a separator for the original graph. We want to remark that Parter and Li are to be credited for this approach, and we simply repeat their main arguments here. To be precise, we merely highlight the *inherent* properties of their construction that are important for us and fill in some crucial details they sketched in [LP19]. We present the construction in Section 9. In the second part, we present a work-efficient PRAM algorithm that can also be simulated in the HYBRID model. This algorithm finds the separator using an Euler tour, which can be easily computed in the PRAM but not so easily in CONGEST. We sketch the construction Section 9.3. In contrast to the CONGEST algorithm, it can essentially be used as is.

9.1 Planar Graphs: Embeddings, Faces, and Augmentations

This section gives a primer on planar graphs and presents several useful properties. We focus on the aspects important to our algorithms in the CONGEST model. In particular, we only provide a heavily simplified introduction to this well-understood graph class. We refer to the well-known textbook by Distel [Die10] or the draft by Klein and Mozes [Kle24] for an in-depth discussion of planar graphs (and their applications).

We begin our short overview with the definition of a planar graph. In fact, there are several equivalent definitions of a planar graph. Perhaps most famously, a planar graph $G = (V, E)$ can be drawn into the Euclidean plane $\mathbb{R}^2$ without two edges crossing. By *drawing*, we mean that nodes are mapped to points and edges to curves/lines between these points. A drawing of a planar graph is called a *geometric planar embedding*. Another way to express such an embedding is a local ordering of the edges. Each node $v \in V$ learns a circular ordering $\pi_v : N_v \rightarrow [1, \mathsf{deg}(v)]$ that assigns each of its neighbors a unique number between 1 and $\mathsf{deg}(v)$. This is called a *combinatorial planar embedding*. W.l.o.g., we denote the neighbors of v as $w_1, \ldots, w_{\mathsf{deg}(v)}$ where $i := \pi_v(w_i)$, i.e., we implicitly assume that the neighbors are ordered according to π_v and the subscript is the position in the ordering. Intuitively, if the edges $\{v, w_1\}, \ldots, \{v, w_{\mathsf{deg}(v)}\}$ are drawn in this order, they do not intersect with other edges. Edmonds has shown that combinatorial and geometric embeddings of planar graphs are equivalent [Edm60]. The equivalence between geometric and combinatorial embeddings becomes clearer when we consider the so-called *faces* of the graph. If a planar graph is drawn without crossing edges, it naturally divides the plane into a set of *regions* enclosed by its edges. These regions are called *faces*. For example, in a grid graph, each cell surrounded by four nodes is a face. Generally, a face is bounded by a closed walk $(v_0, \ldots, v_\ell, v_0) \subseteq V$ in G called the *boundary* of the face. By convention, the unbounded area outside the whole graph is also a face. Given a combinatorial planar embedding (and not a drawing), we can find the walk $(v_0, \ldots, v_\ell, v_0)$ around a given face F by consistently following the *next* edge in the ordering. That means, if we reach a node v via edge $\{w_i, v\}$, we continue via $\{v, w_{i+1}\}$. Note that we omit modulo computations for readability. If a node has degree 1, a walk uses the same edge twice in a row, i.e., the walk enters and leaves via the same edge. A proper embedding ensures that any walk that contains $\{v, w_{i+1}\}$ will eventually return to v via $\{w_i, v\}$. Not that the same node may be visited more than once in a single face walk (this will be important later).

9.2 Fast Separators in CONGEST

In this section, we will prove (the CONGEST part of) Lemma 9.1 and explain how to construct separators on many *arbirtary* node-disjoint subgraphs of a planar graph in parallel in the CONGEST model. To be precise, we are given a collection of node-disjoint subgraphs $\mathcal{C}_1, \ldots, \mathcal{C}_N$ and compute separators that consist of at most four $(1+\epsilon)$-approximate shortest paths in $\tilde{O}(\mathrm{HD})$ time for all them.

Before we go into the details of our construction, let us first summarize the relevant previous approaches to constructing separators in planar graphs in the CONGEST model and emphasize the main technical problems we need to solve: The problem of constructing path separators was (to the best of our knowledge) first considered in [GP17] by Ghaffari and Parter. Their algorithm creates a separator for any collection of **biconnected** disjoint subgraphs in $\tilde{O}(\mathrm{HD})$ time in parallel. In our scenario, however, the subgraphs are *not* necessarily biconnected. So, we cannot use this algorithm directly. A possible approach to extend [GP17] to arbitrary connected subgraphs has already been described by Li and Parter in [LP19]. They compute a separator for a non-biconnected planar graph G. To this end, they first execute an algorithm that adds additional virtual edges to make G biconnected in $\tilde{O}(\mathrm{HD})$ time. The virtual edges are added so that any CONGEST algorithm on the augmented graph G' can be simulated by the original graph G with only a constant factor blowup in the runtime. Then, they run the algorithm of [GP17] on the augmented graph. At first glance, this *should* also result in an algorithm that constructs a separator for a collection of disjoint subgraphs in $\tilde{O}(\mathrm{HD})$ time and, thus, prove Theorem 7. However, there is a subtle yet important issue if we were to apply this approach to a collection of disjoint subgraphs: Consider a collection of node-disjoint connected subgraphs $\mathcal{C}_1, \ldots, \mathcal{C}_N$ of G. Suppose we make the whole graph G biconnected by adding virtual edges (using the algorithm of [LP19]); there may be virtual edges between nodes of two subgraphs $\mathcal{C}_i$ and $\mathcal{C}_j$. These paths between components are required for the biconnectivity. Thus, the extended biconnected subgraphs are not necessarily edge-disjoint anymore, and the algorithm of [GP17] is not (trivially) applicable. If, instead, we apply the algorithm for biconnectivity on each subgraph $\mathcal{C}_i$ in isolation, i.e., by ignoring the edges to other subgraphs, we cannot trivially guarantee that the planarity of the overall graph G is preserved. The virtual edges we add in each subgraph may cross the edges between connected components. However, the algorithm of [GP17] requires planarity of the full graph G if we want to use it in a black-box fashion. So, computing separator in connected node-disjoint subgraphs in $\tilde{O}(\mathrm{HD})$ time requires some extra work.

Li and Parter *claimed* that their algorithm can be refined to $\tilde{O}(\mathrm{HD})$ by exploiting how the algorithm of [GP17] actually works internally and not just using it as black-box. We will follow their sketch for the most part, but we will need to make subtle technical adjustments along the way to make it work. Our main addition to their sketch is adding virtual *buffer nodes* between components. In particular, we add two virtual nodes v' and w' on each edge $\{v, w\} \in E$ where $v \in \mathcal{C}_i$ and $w \notin \mathcal{C}_i$, i.e., we replace the edge by a path (v, v', w', w). We say that in the resulting graph G', all components $\mathcal{C}_i, \ldots, \mathcal{C}_N$ are well-separated as all their neighboring nodes that are *not* assigned to any subset. We add the virtual node v' to $\mathcal{C}_i$ and obtain a component $\mathcal{C}'_i$. Note that these additions a) preserve the graph's planarity, b) increase the hop diameter by a factor of only 3, and c) ensure that $\mathcal{C}'_1, \ldots, \mathcal{C}'_N$ are node-disjoint. Further, any CONGEST algorithm for the augmented graph G' can be simulated on G. We then augment each subgraph $\mathcal{C}'_i$ by additional virtual edges to make it biconnected using the algorithm of Li and Parter [LP19]. Although it might not be intuitively clear, as we will see in the analysis, **the addition of the virtual nodes ensures that the additional edges preserve the planarity of the full graph** G. This results in node-disjoint, biconnected components $\mathcal{C}'_1, \ldots, \mathcal{C}'_N$. However, by adding virtual nodes to the sets $\mathcal{C}_1, \ldots, \mathcal{C}_N$, we can no longer use Ghaffari and Parter's algorithm as a black box because we have changed the input. Instead, we need to adapt the algorithm such that it *ignores* the additional nodes when computing the separator. To be precise, we present an adapted version of the algorithm from [GP17] that works for weighted nodes. Given this algorithm, we assign each virtual node a *weight* of 0 and all other nodes, i.e., the real nodes, a weight of 1. Then, we compute a separator $\mathcal{S}''_i$, such that each connected component $\mathcal{C}''_i \setminus \mathcal{S}''_i$ only has a constant fraction of the total weight. As all virtual nodes have weight 0, the separator $\mathcal{S}''_i$ can be used as a separator for $\mathcal{C}_i$ as desired.

We structure the section into five subsections. First, in Section 9.2.1, we present useful tools for computations on planar graphs in CONGEST. Then, we present three subroutines that our algorithm will use. In Section 9.2.2, we show how to identify biconnected components in a planar graph using the algorithm of Ghaffari and Parter. This section only summarizes their results and adds no new insights, however, we require the algorithmic tools and structural observations used in this algorithm. Then, in Section 9.2.3, we sketch how to turn a planar graph into a biconnected planar graph using a technique by Parter and Li [LP19]. In particular, we show that the algorithm is executed on a series of node-disjoint well-separated subgraph, we can compute biconnected node-disjoint supersets of the subsets. As the last subroutine, in Section 9.2.4, we sketch how to construct a separator for a node-weighted, biconnected planar graph by adapting the corresponding algorithm by Ghaffari and Parter [GP17]. Finally, we put all these pieces together in Section 9.2.6 and present the algorithm behind Lemma 9.1 along with the proof.

9.2.1 A Small Toolkit for Planar Graphs in CONGEST

We now present some useful distributed computing aspects of planar graphs. First, we note that a combinatorial embedding of a planar graph can be computed in $\tilde{O}(\mathrm{HD})$ rounds in CONGEST using the algorithm of Ghaffari and Haeupler [GH16a]. They show that it holds:

Lemma 9.2 (Main Result in [GH16a])**.** *Let $G := (V, E)$ be a planar graph. Then a combinatorial planar embedding of G can be computed in $\tilde{O}(HD)$ rounds in CONGEST. That means each node $v \in V$ learns a local ordering π_v of its neighbors.*

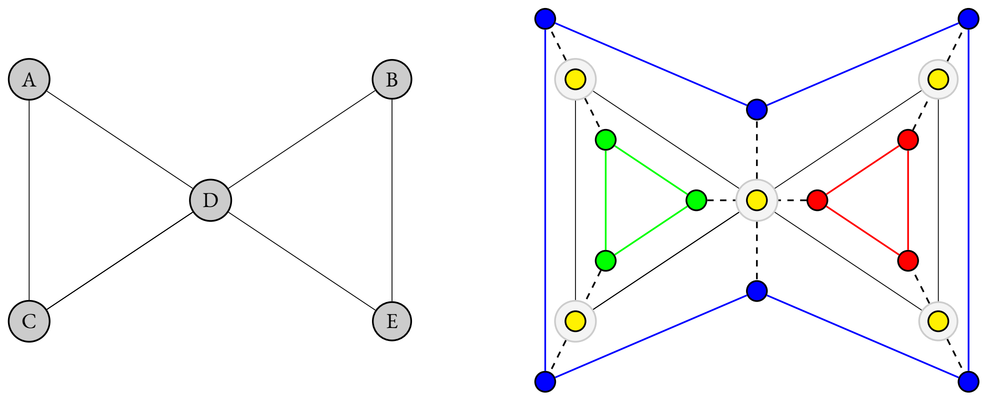
Note that this embedding will allow us to detect the faces of the graph G in a distributed fashion.

For many algorithms, arguing about the graph's faces will be easier than using its nodes and edges. Therefore, we present the so-called face graph $\mathcal{F}(G)$ of a planar graph G, which was introduced in [GP17]. The face graph is a virtual auxiliary graph that simplifies the presentation of our algorithms and proofs. Many parts of the algorithm can be much more easily described in the context of faces rather than vertices and edges. The face graph $\mathcal{F}(G) := (V_S \cup V_F, E_F \cup E_S)$ of a planar graph G consists of virtual nodes and edges, s.t., each boundary is represented by a disjoint set of virtual nodes. In particular, all nodes associated with a given face, form a connected cycle in the face graph. Further, the cycles of two distinct faces are disjoint. To define the face graph, we note that two consecutive neighbors $w_i, w_{i+1} \in N_v$ belong to the face F_i. If v has only a single neighbor, it is only part of one face. Node v creates a virtual node $v^{(i)}$ for each face F_i it is adjacent to. We call these the *face nodes*. Altogether, all face nodes form the set V_F. Face node $v^{(i)}$ is connected to the corresponding virtual nodes of w_i and w_{i+1} that lie on the same face F_i. We call resulting (virtual) edges E_F the **face edges**. Finally, v creates a virtual node v_s that is adjacent to its virtual nodes $v^{(1)}, \dots, v^{(\mathsf{deg}(v))}$. We call v_S the star node of v and denote the set of star nodes as V_S and the resulting edges E_S the **star edges**. Summing up, face graphs are defined as follows:

Definition 9.2 (Face Graph)**.** *For a planar (sub-)graph G the face graph $F_G := (V_F \cup V_S, E_F \cup E_S)$ is a virtual graph defined in the following way:*

1. *For each node $v \in V$ which is on the boundary of faces $F_1, \dots, F_{\mathsf{deg}(v)}$ there are face nodes $v^{(1)}, \dots, v^{\mathsf{deg}(v)} \in V_F$.*
2. *Each face node is connected to at most two face nodes of the same face, i.e, for each face F_i there is a connected cycle of face nodes.*
3. *For each node $v \in V$ there is exactly one star node $v_S \in V_S$ that is connected to all face nodes of $v_i \in V_F$.*

Next, we can turn to the useful computational properties of the face graph. Most importantly, any CONGEST algorithm on the face graph $\mathcal{F}(G)$ can be executed in (asymptotically) the same time as in G. The necessary details to prove this claim were already laid out in [GP17], and we briefly summarize them here to be self-contained. First, we see that we can easily simulate *any* algorithm for F_G on G.



(a) A planar graph G with 5 nodes and 3 faces (including the outer face)

(b) The corresponding face graph $\mathcal{F}(G)$. Star nodes are yellow. Nodes of the same face have the same color.

Figure 9.1: A planar graph G and corresponding face graph $\mathcal{F}(G)$. Note that G is not biconnected. Node D is cut-node that has two of its virtual nodes in the same face as two of its (virtual) nodes have the same color.

Lemma 9.3 (Shown in [GP17]). *Given a planar embedding, any CONGEST algorithm that takes τ rounds on the $\mathcal{F}(G)$ can be simulated on G in 2τ rounds. At all times, a real node $v \in V$ knows the state of its associated star and face nodes.*

Proof. Given a planar embedding of G, the face graph $\mathcal{F}(G)$ can be constructed within $O(1)$ rounds of communication in CONGEST. Fix a node $v \in V$ and let $w_1, \dots, w_{\mathsf{deg}(v)} \in V$ be its neighbors in clockwise order. For each face F_i whose boundary contains the edges $\{v, w_i\}$ and $\{v, w_{i+1}\}$, v creates the face node $v^{(i)}$ with identifier $v \oplus i$. Then, it sends this identifier to w_i and w_{i+1}. Likewise, v receives the identifiers $w_i \oplus j_{w_i}$ and $w_{i+1} \oplus j_{w_{i+1}}$ from w_i and w_{i+1}. Recall that each edge it part of two faces and we need to send two identifiers along each edge. As both identifiers can be encoded in $O(\log n)$ bits, they can be sent within one round of CONGEST.

After completing this preprocessing, the simulation is straightforward. All communication between star nodes and virtual nodes hosted by the same real node can be simulated locally. Further, note that any two neighboring face nodes are hosted by two adjacent real nodes. If a face node $v^{(i)}$ hosted by node v wants to a send a message to virtual node $w^{(j)}$ hosted by real node, it uses the real edges $\{v, w\}$. Thus, we must show that that not too many face node use the same real edge to send their messages. However, this follows directly from the construction of the face graph. Consider an edge $\{v, w_i\}$, then only the face nodes $v^{(i-1)}$ and $v^{(i)}$ use $\{v, w_i\}$ to communicate with their virtual neighbors hosted by node w_i. Thus, within two rounds of CONGEST, the virtual node can send and receive the corresponding messages. This proves the lemma. $\square$

Second, the graph F_G has shortcut quality $\tilde{O}(\mathrm{HD})$. Together with the fact that an CONGEST algorithm on $\mathcal{F}(G)$ can be simulated in G, it holds

Lemma 9.4 (Implied in [GP17]). *Any τ-round* MinAgg *algorithm on a subset of face graphs $\mathcal{F}(\mathcal{C}_1), \dots, \mathcal{F}(\mathcal{C}_N)$ can be simulated in $\tilde{O}(\tau \cdot HD)$ time on G in CONGEST.*

9.2.2 Subroutine 1: Finding and Communicating in Biconnected Components

In this section, we present a distributed algorithm that identifies the biconnected components in (subgraphs of a) planar graph G. Moreover, we will show that we can efficiently perform aggregations in each biconnected component in $\tilde{O}(\mathrm{HD})$ time, w.h.p. Note that the latter is not trivially true despite the graph being planar, as two biconnected components are only edge-disjoint and not node-disjoint. That means they might share a common node node between them. Recall that the minor aggregation framework is only defined with respect to node-disjoint subgraphs. Thus, we require additional arguments based on the topological properties of planar graphs.

Before we get to the main technical lemmas, let us first introduce some notation. We consider a collection of node-disjoint subgraphs $\mathcal{C}_1, \ldots, \mathcal{C}_N$ of a planar graph G with hop-diameter HD. The cut-nodes $\mathfrak{C}_i \subset V_i$ are nodes of $\mathcal{C}_i$ whose removal increases the number of connected components in $\mathcal{C}_i$ by at least one. In other words, as $\mathcal{C}_i$ is connected, the removal of a cut-node $u \in \mathfrak{C}_i$ disconnects $\mathcal{C}_i$. Conversely, all other nodes $v \in V_i \setminus \mathfrak{C}_i$ must be part of at least one cycle in $\mathcal{C}_i$. We call a subgraph $B_{i_j} \subset \mathcal{C}_i$ a biconnected component, or simply a *block*, if and only if for any two nodes $v, w \in B_{i_j}$ there are (at least) two node-disjoint paths between v and w in $\mathcal{C}_i$. We refer to the set of all blocks in $\mathcal{C}_i$ as $\mathfrak{B}_i$. Note that a cut-node $v \in \mathfrak{C}_i$ may be part of one or more blocks, but each edge $\{v, w\}$ is associated with exactly one block.

First, we want to compute the cut-nodes and blocks of a collection of node-disjoint subgraphs $\mathcal{C}_1, \ldots, \mathcal{C}_N$ of a planar graph G in $\tilde{O}(\mathrm{HD})$ time. In particular, each cut-node will learn that it is a cut-node and which of its neighbors belong to the same biconnected component. All other nodes learn that they are *not* cut-nodes and that therefore all their neighbors belong to the same biconnected component. We will show that it holds:

Lemma 9.5 (Identification Biconnected Components). *Let $G := (V, E)$ be planar graph and let $\mathcal{C}_1, \ldots, \mathcal{C}_N$ be a collection of node-disjoint subgraphs of G. Then, there is an algorithm that computes all cut-nodes and biconnected components in all $\mathcal{C}_1, \ldots, \mathcal{C}_N$ in $\tilde{O}(HD)$ rounds in CONGEST, w.h.p.*

The problem of finding the cut-nodes in the whole graph G is well understood due to an algorithm by Ghaffari and Parter [GP17]. Their algorithm identifies all biconnected components of a planar graph G in time $\tilde{O}(\mathrm{HD})$, w.h.p, and can be applied to subgraphs $\mathcal{C}_1, \ldots, \mathcal{C}_N$. Ghaffari and Parter exploit a very interesting relationship between the face graph and biconnectivity. In planar graphs, it holds:

Lemma 9.6 (Cut-nodes via Faces, Section C.1 in [GP17]). *A node $v \in V$ in a planar graph $G = (V, E)$ is a cut-node **if and only if** in its face graph $\mathcal{F}(G)$, two or more of its corresponding face nodes belong to the same face.*

Proof. First, suppose no two face nodes of v belong to the same face. In this case, we must show that for any two neighbors w_i, w_j of v there is a path from w_i to w_j that does not contain v. Recall that the edges $\{v, w_i\}$ and $\{v, w_{i+1}\}$ to two consecutive neighbors w_i and w_{i+1} in the embedding lie on the walk around a face F_i. If no two face nodes lie on the same face, the walk around F_i starts via edge $\{v, w_i\}$ and returns via $\{v, w_{i+1}\}$ without visting v again. Thus, there is a path from w_i to w_{i+1} that does not contain v. By the same argument, there is a path between w_{i+1} and w_{i+2} and so on. Via a simple induction, we conclude that there is a path between any two neighbors w_i and w_j that does not contain v. This proves the first part.

Now suppose there are two face nodes of v that belong to the same face F. In this case, v must be a cut-node and the removal of v disconnects the graph. If two face nodes of v belong to the same face, this means that v is visited (at least) twice the walk around the face. W.l.o.g. consider the walk $(v, v_1, \ldots, v_\ell, v, v_{\ell+1})$. In other words, we start the start the walk at node v and $v_{\ell+1}$ is the first node we visit after returning to v. Recall that we can draw a face F by starting at v and always draw the next edge in clockwise order. Thus, all edges of nodes v_1 to v_ℓ must be on the *inside* of face F. If there is path from any node v_1 to v_ℓ that does contain v then there is a path that goes from inside the face F outside of it without containing node on the boundary. Therefore, it must cross the boundary which it cannot do since the graph is planar. Thus, node $v_{\ell+1}$ is isolated from the remaing nodes on the face, which makes v a cut-node as desired. □

Figure 9.1 shows an example where the lemma can be verified visually: The central node has two of its virtual nodes on the outer face. All other nodes have virtual nodes in distinct faces, these nodes may be removed without disconnecting the graph. Given this lemma, we can prove Lemma 9.5.

Proof of Lemma 9.5. First, we compute the face graph $\mathcal{F}(G)$ of G and then delete virtual nodes and edges associated with edges between components. We obtain a face graph $\mathcal{F}(\mathcal{C}_i)$ for each $\mathcal{C}_i$. Note that all these face graphs are still planar, but may have a diameter larger than $3 \cdot \text{HD}$. By Lemma 9.6, each cut-node in $\mathcal{C}_i$ has two virtual nodes in the same face in the corresponding face graph in $\mathcal{F}(\mathcal{C}_i)$. To identify the faces, we elect a leader in each face component. If two face nodes have the same leader, the corresponding star node can locally decide if it is a cut-node. A leader can be elected in $\widetilde{O}(\text{HD})$ time by the following subroutine:

1. Aggregate the minimal identifier on each face component simultaneously in $\widetilde{O}(\text{HD})$ time. This is possible because the face components of each $\mathcal{F}(\mathcal{C}_i)$ are still node disjoint subgraphs of $\mathcal{F}(G)$ (per definition of the face graph), and any aggregation on node disjoint sets can be performed in time $\widetilde{O}(\text{HD})$ as per Lemma 9.4.

2. Each face node shares this identifier with its respective star node. This can be done locally, as the same real node simulates all face nodes in question.

3. If a star node v_S receives the same identifier from two or more neighboring face nodes, the corresponding node $v \in V$ is a cut-node. This follows from Lemma 9.6.

Summing up, it holds that all cut-nodes can be simultaneously identified in $\widetilde{O}(\text{HD})$ time. Next, consider the blocks $\mathfrak{B}_i$. Each node knows which of its edges belong to the same block. For a cut-node, these are edges between two edges of the same face (according to its local ordering); for all other nodes, it's simply all their edges. Thus, after $\widetilde{O}(\text{HD})$ time, w.h.p., each node knows whether it is a cut-node and which of its edges belong to the same block. This proves Lemma 9.5. □

Second, we want to show that we can compute an aggregation function $\bigoplus$ for each biconnected component in $\widetilde{O}(\text{HD})$, w.h.p. That is, each node $v \in B_{i_j}$ which is part of a biconnected component B_{i_j} chooses an $O(\log n)$-bit input value x_v. Then, all nodes in B_{i_j} learn the aggregate value $\bigoplus_{v \in B_{i_j}} x_v$. In particular, a cut-node that (by definition) is in at least two biconnected components learns the aggregate value for all of them. Formally, we want to show the following:

Lemma 9.7 (Communication in Biconnected Components). *Let $G := (V, E)$ be planar graph and $\mathcal{C}_1, \dots, \mathcal{C}_N$ be a collection of node-disjoint subgraphs of G. For each component $\mathcal{C}_i$, let $\mathfrak{B}_i := \{B_{i_1}, \dots, B_{i_{N_i}}\}$ be the biconnected components of $\mathcal{C}_i$. Suppose, each node $v \in B_{i_j}$ chooses a $\tilde{O}(1)$-bit value x_v. For each biconnected component B_{i_j}, we define $y_{i_j} := \bigoplus_{v \in B_{i_j}} x_v$, where $\bigoplus$ is any pre-defined admissible aggregation operator. Then, there is an algorithm that in **all** biconnected components B_{i_j}, lets all nodes $v \in B_{i_j}$ learn y_{i_j} in $\tilde{O}(HD)$ time, w.h.p.*

Proof. For the proof, we consider the face graph $\mathcal{F}(G)$ of the full graph G and augment it to a planar graph $\mathcal{F}'(G)$. $\mathcal{F}'(G)$ has the same diameter as $\mathcal{F}(G)$ and for all biconnected components in $\mathcal{C}_1, \dots, \mathcal{C}_N$ there are connected node-disjoint components of $\mathcal{F}'(G)$. In particuluar, if a biconnected component contains a node $v \in V$, the connected component in $\mathcal{F}'(G)$ contains at least one virtual node simulated by G. Therefore, any message received by this node will be received by v.

The augmentation works as follows: Let $v \in V$ be a cut-node in its respective component $\mathcal{C}_i$ and let v_s be its star node and $v^{(1)}, \dots, v^{(\mathsf{deg}(v))}$ its virtual nodes in $\mathcal{F}(G)$. Since $v \in V$ is a cut-node, there are virtual nodes $\{v^{(i_1)}, \dots v^{(i_b)}\}$ such each pair $v^{(i_j)}, v^{(i_{j+1})}$ belongs to the same face of $\mathcal{C}_i$. For any consecutive pair $v_{i_j}, v_{i_{j+1}}$, we connect the virtual nodes $v_{i_j+1}, \dots, v_{i_{j+1}-1}$ in-between by a virtual edge. Note that this includes virtual that are part of the same biconnected component in $\mathcal{C}_i$ as well as some additional nodes not included in $\mathcal{F}(\mathcal{C}_i)$. For each biconnected component B_{i_j} there is a node-disjoint subset in the augmented graph $\mathcal{F}'(G)$ as desired.

As newly connected nodes are consecutive in the embedding, the additional edges preserve the planarity. Further, we only add virtual edges between virtual nodes that belong to the same node in G. Therefore, any CONGEST algorithm on the augmented graph can be simulated without increasing the runtime. Finally, the additional edges can only reduce the diameter. Thus, as $\mathcal{F}'(G)$ is planar and has the same diameter as $\mathcal{F}(G)$, we can perform a minor aggregation of each block in $\tilde{O}(\text{HD})$ time, w.h.p. Therefore, we can compute an aggregation function $\bigoplus$ on each block B_{i_j} in $\tilde{O}(\text{HD})$ time, w.h.p. This proves Lemma 9.7. □

9.2.3 Subroutine 2: Making Planar Graphs Biconnected

In this section, we show a technical auxiliary result that can turn connected subgraphs of a planar graph into biconnected components. The algorithm works by adding additional virtual edges to each subgraph. These edges ensure that the selected subgraphs become biconnected while the overall graph stays planar. Further, the edges are added in such a way that they can be simulated in a CONGEST algorithm without slowing down the execution too much.

The underlying algorithm was (for the most part) already presented in [LP19] by Li and Parter. They considered the problem of making a single planar subgraph $G' \subset G$ biconnected by adding virtual edges. However, the algorithm does not (trivially) work on any collection of connected subgraphs because of the reasons we mentioned in the introduction of this section: If we independently add virtual edges to each component, this may violate the planarity of graph G. Therefore, we will focus on what we call *well-separated* subgraphs. We call a set of node-disjoint subgraphs $\mathcal{C}_1, \dots, \mathcal{C}_N$ *well-separated* if for two nodes $v \in \mathcal{C}_i$ and $w \in \mathcal{C}_j$ (with $i \neq j$) there are at least two nodes not associated with any subgraph between v and w. This can, for example, be achieved by adding two virtual nodes on the edges between two subgraphs. To put it simply, between every two subgraphs $\mathcal{C}_i$ and $\mathcal{C}_j$, there are two *buffer* nodes not associated with any subgraph. This node will give us the necessary freedom to add the virtual edges without violating planarity and keeping the components disjoint. For any connected subgraph $\mathcal{C}_i := (V_i, E_i)$, we define

$$V_i' := \{v \in V \mid \exists w \in V_i : \exists\{v, w\} \in E\}\} \tag{9.1}$$

The set V_i' contains all nodes of $\mathcal{C}_i$ and all nodes adjacent to a node in $\mathcal{C}_i$. As the subgraphs $\mathcal{C}_1, \dots, \mathcal{C}_N$ are well-separated, all sets $V_1', \dots, V_N'$ are disjoint. Throughout this section, we will consider the subgraphs $\mathcal{C}_1', \dots, \mathcal{C}_N'$ where $\mathcal{C}_i' = G[V_i']$ is the subgraph induced by V_i'. Note that all these subgraphs are node-disjoint and $\mathcal{C}_i \subset \mathcal{C}_i'$. Given these definitions and preliminary thoughts, we will show the following lemma in the remainder of this section:

Lemma 9.8 (Derived from [LP19]). *Let $G = (V, E)$ be a planar graph and let $\mathcal{C}_1, \dots, \mathcal{C}_N$ be a set of N node-disjoint, well-separated subgraphs of G. Then, within $\tilde{O}(HD)$ time in CONGEST, we can compute a graph $\widehat{G}$ such that*

1. *$\widehat{G}$ is still planar and has diameter HD.*
2. *For each subgraph $\mathcal{C}_i$, there is a biconnected subgraph $\widehat{\mathcal{C}_i} \supset \mathcal{C}_i$. The subgraphs $\widehat{\mathcal{C}_1}, \dots, \widehat{\mathcal{C}_N}$ are edge-disjoint.*
3. *Any CONGEST algorithm on $\widehat{G}$ that takes τ rounds can be simulated in $\tilde{O}(\tau)$ rounds on G.*

Let us begin by presenting Li and Parters' approach to making any planar graph biconnected. The high-level idea of the construction is that each cut-node creates a *cycle* of virtual edges that connect its neighbors. In other words, it creates a path between all its neighbors that does not contain itself. While this sounds easy, one must ensure that the resulting graph is still planar. We further require the concept of a *block-cut tree*. A block-cut tree $\mathfrak{T}(G) := (\mathfrak{B}_G \cup \mathfrak{C}_G, E_{BC})$ represents the biconnected components in a given (planar) graph $G = (V, E)$. The tree has two types of nodes: *cut* nodes $\mathfrak{C}_G$ and *block* nodes $\mathfrak{B}_G$. In particular, the blocks are supernodes represented by connected edge-disjoint subgraphs. Note that a cut-node may appear in several blocks. The edges of $\mathfrak{T}(G)$ are defined as follows: Each cut-node has an edge to each block node that contains it. It is easy to verify that this definition can only result in a tree.

Given the definition a block-cut tree, Li and Parters' algorithm works as follows: We begin by computing the block-cut tree of each component $\mathcal{C}_i$ and rooting it in a cut-node $r_i \in \mathfrak{C}_i$. Through rooting the tree, each cut-node (except the root) obtains a unique parent-block and a number of child-blocks. We connect the parent- and child-blocks of each cut-node in two steps: First, every cut-node connects any two consecutive neighbors in its child-blocks. This merges all its child-blocks into one. We refer to the virtual edges added in this step as the A_i-edges. Thus, after adding the A_i-edges, each cut-node has at most one parent-block and at most one child-block. Second, each cut-node connects its parent- and child-block by a virtual edge. To prevent the virtual edges from crossing (and therefore violating the planarity), we employ two different rules: All cut-nodes $v \in \mathfrak{C}_i$ determine the number $\ell(v)$ of cut-nodes on their path to r_i in the block-cut tree $\mathfrak{T}_i$. We call $\ell(v)$ the level of v. All cut-nodes with *even* level connect two consecutive neighbors w_i and w_{i+1} if and only if w_i is in the parent-block and w_{i+1} is in the child-block. Conversely, all cut-nodes with *odd* level connect two consecutive neighbors w_i and w_{i+1} if and only if w_i is in the child-block and w_{i+1} is in the parent-block. In other words, nodes on even levels connect parents to children in clockwise order, and nodes on odd levels connect them in counterclockwise order. We refer to the virtual edges added in this step as the B_i-edges. Figure 9.2 presents a detailed description of all steps.

```
MakeBiconnected($\mathcal{C}'_i$):

(S1) Compute all cut-nodes $\mathfrak{C}_i$ and blocks $\mathfrak{B}_i$ in $\mathcal{C}'_i$. Construct a block-cut tree $\mathfrak{T}_i$ of $\mathcal{C}'_i$ rooted in
     cut-node $r_i \in \mathfrak{C}_i$ with highest identifier.

(S2) For each cut-node $v \in \mathfrak{C}_i$ and block $B_{i_j} \in \mathfrak{B}_i$, compute the number $\ell(\cdot)$ of cut-nodes on the
     unique tree path to $r_i$ in $\mathfrak{T}_i$. We call $\ell(\cdot)$ the level of a node. Set the level of each non-cut-node
     $u \in B_{i_j}$ and edge $\{u, w\} \in B_{i_j}$ to $\ell(B_{i_j})$.

(S3) Every cut-node $v \in \mathfrak{C}_i$ in level $\ell(v)$ connects every two consecutive neighbors $w_i, w_{i+1}$ in the
     clockwise ordering satisfying that

         $\ell(\{v, w_i\}) = \ell(\{v, w_{i+1}\}) = \ell(v) + 1$

     by a virtual edge. That is, the cut-node $v$ connects consecutive children in the block-cut tree. We
     denote the set of (virtual) edges this process creates as $A_i$.

(S4) Compute a planar embedding of $\mathcal{C}^A_i = (V'_i, E'_i \cup A_i)$. Set the level of each $\{w_i, w_{i+1}\}$ added
     by $v \in \mathfrak{C}_i$ to $\ell(v) + 1$.

(S5) For each cut-node $v \in \mathfrak{C}'_i$ on an even level (where $\ell(v) \mod 2 = 0$), we add the following
     (virtual) edge $\{w_i, w_{i+1}\}$ between consecutive nodes $w_i$ and $w_{i+1}$ if it holds:

         $\ell(\{v, w_i\}) = \ell(v) + 1 \wedge \ell(\{v, w_{i+1}\}) = \ell(v)$

     We denote the resulting set of edges as $B^{even}_i$. The cut-nodes $v \in \mathfrak{C}'_i$ on odd (where
     $\ell(v) \mod 2 = 0$) levels add a (virtual) edge $\{w_i, w_{i+1}\}$ between consecutive nodes $w_i$ and
     $w_{i+1}$ if it holds:

         $\ell(\{v, w_i\}) = \ell(v) \wedge \ell(\{v, w_{i+1}\}) = \ell(v) + 1$

     We denote the resulting set of edges as $B^{odd}_i$.

(S6) Return $\widehat{\mathcal{C}}_i := (V'_i, E'_i \cup A_i \cup B^{even}_i \cup B^{odd}_i)$
```

Figure 9.2: Pseudocode for the MakeBiconnected algorithm of Li and Parter [LP19] that turns any planar graph into a biconnected planar graph by adding additional (virtual) edges to each node.

To prove Lemma 9.8, we must show that each resulting component is biconnected, the overall graph remains planar, and any CONGEST algorithm can be be simulated with constant multplicative overhead. Further, we must show that the algorithm that computes the additional edges can be implemented in $\widetilde{O}(\mathrm{HD})$ time.

First, we show that the algorithm lives up to its name and ensures that augmented components are biconnected. This proof follows from the algorithm more or less straightforwardly. The algorithm creates a path of virtual edges around each cut-node that connects all its neighbors. Thereby, it merges as biconnected components into one. Formally, it holds:

Lemma 9.9 ($\widehat{C}_i$ is biconnected)**.** *Let $\widehat{C}_i = (V_i', E_i \cup A_i \cup B_i)$ be the subgraph that results from executing* $\textit{MakeBiconnected}(\mathcal{C}_i')$*. Then, $\widehat{\mathcal{C}}_i$ is biconnected.*

Proof. For the proof, we consider a cut-node $v \in \mathfrak{C}_i$ with neighbors $w_1, \ldots, w_{\mathsf{deg}(v)}$. We show that for any two neighbors w_j and $w_{j'}$, there is a path $P_{w_j w_{j'}}$ connecting these two nodes that does not contain v. This is sufficient to show biconnectivity. To see this, suppose we remove v from $\widehat{C}_i$. Then, any path that contains the edges $\{w_j, v\}$ and $\{v, w_{j'}\}$ can be *rerouted* via $P_{w_j w_{j'}}$ and $\widehat{C}_i \setminus \{v\}$ stays connected.

Now consider two neighbors w_j and $w_{j'}$. Note that we can assume that w_j and $w_{j'}$ are in different blocks (as otherwise there already would be a path between w_j and $w_{j'}$ and there is nothing to prove). Given this conditioning, we prove the existence of the path $P_{w_j w_{j'}}$ in two batches. First, we show that there is a path between any w_j and $w_{j'}$ in two different child-blocks of v. Recall that all neighbors in the parent-block are consecutive in the embedding. W.l.o.g. assume that these are the first $\delta \leq \mathsf{deg}(v)$ nodes $w_1, \ldots, w_\delta$ in the ordering. Thus, all other nodes are on a lower level. Then, in step (S3) in Algorithm 9.2, we add the edges $\{w_{\delta+1}, w_{\delta+2}\}, \ldots, \{w_{\mathsf{deg}(v)-1}, w_{\mathsf{deg}(v)}\}$. Therefore, all neighbors in child-blocks are connected in $\mathcal{C}_i^A$ by a path of the newly added edges. Second, we consider two nodes w_j and $w_{j'}$ in a child- and a parent-block. As all nodes in child-blocks belong to the same block in $\mathcal{C}_i^A$, it sufficient to show that (at least) one node in a child-block is connected to (at least) one node in a parent-block. Such an edge is explicitly added in step (S5) in Algorithm 9.2. Thus, for any two neighbors, there is a path of virtual edges, and $\widehat{C}_i$ is biconnected. ☐

We continue to show that two subgraphs $\widehat{C}_i$ and $\widehat{C}_j$ are edge-disjoint. For this proof, we exploit the fact that the subgraphs $\mathcal{C}_1, \ldots, \mathcal{C}_N$ are well-separated. The *buffer* nodes cannot be cut nodes and therefore do not add edges between compoents. It holds:

Lemma 9.10. *Let $\widehat{C}_i = (V_i', E_i \cup A_i \cup B_i)$ be the subgraph that results from executing* $\textit{MakeBiconnected}(\mathcal{C}_i')$*. Then, any two subgraphs $\hat{C}_i$ and $\hat{C}_j$ are edge-disjoint.*

Proof. Recall that only cut-nodes add new edges between their neighbors to the graph. This holds for both the A-edges and the B-edges. Therefore, the only a cut-node with respect to $\mathcal{C}_i$ that has a neighboring node in $\mathcal{C}_j$ (or vice versa) can add an edge between components. For this proof, we focus on the nodes of $\mathcal{C}_i'$ that may add an edge. The other case is analogous. By construction, the only nodes that may have neighbors in a different component are the *buffer* nodes we added to $\mathcal{C}_i$ obtain $\mathcal{C}_i'$. Thus, we must show that these *buffer* nodes cannot be cut-nodes (with respect to $\mathcal{C}_i'$). As we added them to a connected subgraph $\mathcal{C}_i$, for any of their neighbors in $\mathcal{C}_i$, there must be a path connecting them in $\mathcal{C}_i$. Therefore, they cannot be cut-nodes as upon their removal, there will always be a path connecting their neighbors. This proves the lemma. ☐

We continue with a useful observation on augmenting planar graphs. In many cases, we can simplify algorithms for planar graphs by adding virtual edges. However, in doing so, we must ensure that these edges do not violate the planarity. In sequential algorithms, this can often be done by adding edges sequentially making sure that each new edge does cross any of the previously added edge. This, of course, is infeasible in a distributed algorithm. Here, we need to add many edges at once in order to keep the running time low. However, if we add edges concurrently, we need to be careful. To this end, we prove the following claim that identifies which edges are *safe* to add concurrently. Note that this claim was implicitly used but not proven in [LP19]. Thus, we close the gap left in that paper by explicitly showing it. It holds:

Claim 12. *Suppose a node $v \in V$ creates an edge $\{w_i, w_{i+1}\}$ between two neighboring nodes w_i and w_{i+1} in its embedding. Let $W_F = (x, w_i, v, w_{i+1}, y, \dots)$ be a face walk that contains edges $\{w_i, v\}$ and $\{v, w_{i+1}\}$. Then, the only edges that may intersect with $\{w_i, w_{i+1}\}$ are $\{x, v\}$ and $\{v, y\}$.*

Proof. We prove the lemma as follows: Let E' be a set of edges we want to (simultaneously) add to graph G. For each edge $\{u, w\} \in E'$, there is a node $v \in V$ such that u and w are consecutive in its embedding. Let $W_F = (x, u, v, w, y, \dots)$ be a face walk that contains edges $\{u, v\}$ and $\{v, w\}$. In other words, x precedes v in u's and v precedes y in w's embedding. To prove the lemma, we will show that *if* E' does not contain $\{x, v\}$ are $\{v, y\}$, all edges can be safely added without violating planarity. This then proves that all edges that *potentially* can cross edges E'.

Our proof works by induction: We fix an arbitrary order $e_1, e_2, \dots$ of the edge in E' and consider one edge at a time. Our induction hypothesis is that the graph $G_i := (V, E \cup \{e_1, \dots, e_i\})$ is planar and for each edge $\{u_j, w_j\} \in E' \setminus \{e_1, \dots, e_i\}$, it holds that u_j and w_j are still consecutive in the embedding of some node v_j. The latter argument is important as it implies that all edges can be added in parallel.

Initially, before we add any edges, the hypothesis is trivially true. For the induction step, suppose v_i adds an edge $e_i := \{u_i, w_i\}$ between u_i and w_i. Now consider the face walk on a face F_i that contains edges $\{u_i, v_i\}$ and $\{v_i, w_i\}$

$$W_{F_i} := (v_i, w_i, y_i, \dots, x_i, u_i, v_i). \tag{9.2}$$

Here, x_i is the hop the walk takes before u_i, and y_i is the hop after w_i. In the embedding of u_i, we place the new edge to w_i between v and x. Analogously, in the embedding of w_i, place the new edge to u_i between y_i and v_i.

First, let us argue why this preserves the planarity: Adding the edge creates the two new faces F_i' and F_i'' with valid face walks: First, the small face F_i' with nodes v_i, u_i, and w_i on its boundary. The corresponding walk is

$$W_{F_i'} := (v_i, u_i, w_i, v_i). \tag{9.3}$$

Second, a (possibly larger) face F_i'' that has all nodes of F_i except v_i on it boundary. The corresponding walk $W_{F_i''}$ is almost equal to W_{F_i} except it skips v_i and goes from u_i to w_i directly, i.e., we have

$$W_{F_i''} := (w_i, y_i, \dots, x_i, u_i) \tag{9.4}$$

It remains to show that we prevent no future edges from being added by the same rule. Note that we only change the embedding of u_i and w_i. In the following, we focus on u_i, the other case is analogous. In u_i's case, its neighbors x_i and v_i are no longer consecutive. However, if $\{x_i, u_i\} \in E$, u_i will never add an edge $\{x_i, v_i\}$ by assumption. If edge $\{x_i, u_i\}$ was added in earlier step, then E' does not contain any edge $\{x_i, u_i\}$ as u_i only adds edges that involve its initial neighbors. $\square$

This claim shows that the possible intersections are, in a sense, very local. In particular, the claim directly implies that any two edges added by non-neighboring nodes will not intersect.

Next, we show that the additional edges do not affect the planarity of the overall graph. For this proof, we again exploit that the initial components are well-separated. Therefore, all edges that can potentially be crossed (even those outside of the component) are considered when deciding which edge to add. It holds:

Lemma 9.11 ($\widehat{G}$ is planar). *Let* $\widehat{G} = (V', E \cup \{A_1, B_1, \dots, A_N, B_N\})$ *be the graph that results from executing* MAKEBICONNECTED *on all well-separated, node-disjoint components* $\mathcal{C}'_1, \dots, \mathcal{C}'_N$*. Then,* $\widehat{G}$ *is planar.*

Proof. The main difficulty in proving the lemma is that we add many edges in parallel, which might intersect with one another. Recall that by Claim 12, only edges added by neighboring nodes may intersect. Thus, for the main part of this proof, we consider two neighboring nodes $u \in V$ and $v \in V$ with $\{u, v\} \in E$. W.l.o.g. both u and v are cut nodes. We need to argue that the edges added by u and v do not intersect. Recall that the virtual edges in $\mathcal{C}'_i$ are added in two batches: A_i and B_i. We will consider them one after the other and begin with the A_i-edges. For the purpose of this proof, we give an edge in $\mathcal{C}'_i$ an orientation. A cut-vertex v on level $\ell(v)$ orients its incident edges $\{v, w\}$ in level $\ell(\{v, w\}) = \ell(v) + 1$ *away* from it. Recall that these are the edges in a block that is only reachable from the root via v, i.e., they are in a child-block of v. An edge gets a unique orientation as any other cut-node w that could orient the edge must be on level $\ell(v) + 1$. Note that some edges will not be directed, namely edges between two non-cut-nodes but this does not affect the proof. Reviewing the algorithm under consideration of this orientation, the A_i-edges are added according to the following rule: A cut-node v connects any two *outgoing* neighbors w_i and w_{i+1} via an edge $\{w_i, w_{i+1}\}$. By Claim 12, the only edges that potentially crosses $\{w_i, w_{i+1}\}$ are either $\{x, v\}$ and $\{v, y\}$ where x and y are predecessor of w_i and the successor of w_{i+1} in a face walk W_F respectively. Thus, it must either hold $u = w_i$ or $u = w_{i+1}$ for one or more faces. Assume w.l.o.g. that $u = w_i$. The other case is analogous. Thus, if a node u adds an edge $\{x, v\}$, both x and v must be in a child-block of u per definition. However, if v cannot be in a child block of u because we assumed u is in child-block of v. Therefore, u will not add edge $\{x, v\}$. As the same argument holds for every face, we conclude that the A-edges do not intersect.

Next, we consider the B_i-edges. Again, assume that v adds an edge $\{w_i, w_{i+1}\}$ between consecutive nodes and let $u = w_i$. Then, again only an edge $\{x, v\}$ where x is the predecessor of u on some face walk W_F may violate planarity. Recall that x must be the predecessor of v in u's embedding. Here, we distinguish between two cases based on the levels of u and v:

1. If $\ell(v) = \ell(u)$, the edge $\{u, v\}$ must be part of a block with a cut-node on level $\ell(v) - 1$. Otherwise, if the edge would be between two blocks, it would either close a cycle *or* w_{i+1} would be only reachable via v from the root (or vice versa). In the latter case, $\ell(v)$ and $\ell(w_{i+1})$ must differ by one. Thus, the level of $\{u, v\}$ would be $\ell(u)$ (and not $\ell(u) + 1$).

   Now suppose that the level is even, the other case is analogous. In this case, v would add an edge between parent- and child-block in clockwise order, and w would add an edge between parent- and child-block in counterclockwise order. As both nodes are on the same level, they must use the same ordering. We distinguish between two subcases:

   (a) If x is in child-block of u, the edge $\{x, v\}$ would not be added because we would connect a child to a parent in clockwise order. This is only done by nodes on odd levels.

   (b) If x is a parent-block of u, then u must be the closest node to the root in the block that contains u and v. However, in this case u and v cannot have the same level.

2. If $\ell(u) \neq \ell(v)$, it must hold that difference of the levels is exactly 1. Assume w.l.o.g that $\ell(u) = \ell(v) - 1$, i.e., u is be the closest node to the root in the block that contains u and v. Then, the edge $\{u, v\}$ has level $\ell(v)$, i.e., it is in the child-block of u and the parent-block of v. Now suppose that $\ell(u)$ is odd, which means that $\ell(v)$ is even.

   (a) If x is in child-block of u, the edge $\{x, v\}$ would not be added because we would connect a node from a child-block to another node from a child-block. This is against the rules.

   (b) If x is in the parent-block of u, then u does not add the edge $\{x, v\}$ because odd nodes do not connect parent-block to child-blocks in clockwise order. Only if the x is also successor of v in the embedding of u (which only happens if u's degree is 2), the edge may added. However, in this case, the edge $\{x, v\}$ is added on another face walk $W_{F'}$. Recall that we assumed that v added the edge $\{w_i, w_{i+1}\}$ where $u = w_i$. On $W_{F'}$, the edge $\{u, w_{i+1}\}$ is not added by v. Either w_{i+1} is not part of that face, or if it is, it also appears in a different order. Thus, in either case, it would not be added.

Thus, all in all, $\widehat{G}$ must be planar. □

We conclude the analysis by showing that any CONGEST algorithm can be simulated on $\widehat{G}$. This enables us to execute our algorithms directly on $\widehat{G}$ without needing to worry about how exactly the virtual edges are added. It holds:

Lemma 9.12 ($\widehat{G}$ can be simulated). *Any τ-round CONGEST algorithm on $\widehat{G}$ can be simulated in $O(\tau)$ rounds on G.*

Proof. Let G_A be the graph that results from adding the A-edges, and let $\widehat{G}$ be the graph that results from adding the B-edges to G_A. We prove the lemma in two steps. First, we show that any τ-round CONGEST algorithm on G_A can be simulated in 2τ rounds on G. This follows from the fact that each edge $\{w_i, w_{i+1}\}$ in A corresponds to a path $w_i - v - w_{i+1}$ for some cut-node v in G. All messages intended to be sent along that edge from w_i to w_{i+1} will be sent first to the responsible neighbor v and then to the target. However, we must also ensure that not too many messages are sent along an edge. Recall that in CONGEST, we can only send $O(\log n)$ bits per edge. Therefore, if an edge $\{w_i, v\}$ is responsible for too many virtual edges To this end, note that each neighbor can only be *responsible* for two virtual edges; One edge for each face the edge $\{w_i, v\}$ is part of. Two messages of size $O(\log n)$ still have size $O(\log n)$ if we combine them. Thus, we can simulate both edges in two rounds.

For the second step, recall that the B-edges are added according to the same rule, namely that only consecutive neighbors are connected. Thus, by the same argument, each edge of G_A is responsible for at most two virtual edges. Thus, each τ-round algorithm on $\widehat{G}$ can be implemented in 4τ rounds on G_A. Note that this doubles the message size again, but we are still within $O(\log n)$ Thus, altogether, the simulation of a τ-round algorithm takes $2 \cdot 2 \cdot \tau = 4\tau$ rounds.

This proves the lemma. $\square$

First, we show that the algorithm can be executed in $\widetilde{O}(\text{HD})$ time, w.h.p. We exploit the fact that the algorithm's central computations, i.e., building a block-cut tree and assigning levels to the cut-nodes, can be broken down into aggregation on biconnected components. We show:

Lemma 9.13. *MakeBiconnected can be executed on all $\mathcal{C}'_i, \dots, \mathcal{C}'_N$ in $\widetilde{O}(HD)$ time, w.h.p.*

Proof. We prove the runtime for each step of the algorithm individually.

Step (S1) The cut-nodes and block nodes of all well-separated components can be determined in $\widetilde{O}(\text{HD})$ time, w.h.p, using the algorithm from Lemma 9.8.

Step (S2) If we know the cut-nodes and blocks, the computation of the block-cut tree in each component $\mathcal{C}'_i$ only consists of minor aggregations and works as follows:

1. First, we aggregate the identifiers of all cut-nodes to determine the cut-node $r_i \in \mathfrak{C}_i$ with the minimal identifier in each $\mathcal{C}'_i$. This requires one aggregation.

2. Recall that that we have a spanning tree T_i of each $\mathcal{C}'_i$[1]. We root this tree in r_i using the tree operations of Lemma 7.10. To be precise, we need to compute the AncestorSum for v_0 each node where each cut-node inputs 1. This lets each node learn its hop distance to r_i in T_i. The unique neighbor with higher hop distance is a node's parent with respect to r_i.

3. Each cut-node $v \in \mathfrak{C}_i$ (except the root) has a directed edge to some parent node p_v. This edge must belong so some block $B_{i_j} \in \mathfrak{B}_i$. Recall that even an edge between two cut-nodes belongs to a trivial block. On each tree-edge $e \in T_i$ that leads to a node in a block $B_{i_{j'}} \neq B_{i_j}$, we create a virtual node v_e. Further, we replace edge $e = \{v, w\}$ with the path $v - v_e - w$. Both new edges on this path are added to the tree T_i, so T_i continues to be a spanning tree. The root creates the virtual nodes and edges for *all* its incident edges. These virtual nodes and edges are only temporary and we remove them again, once we are done computing the levels.

4. For each node $v \in V_i$, we then compute the number $\ell(v)$ of virtual nodes on its path to r_i in T'_i. We use the resulting number $\ell(v)$ as the level of v. This can be done via the Tree Aggregation from Lemma 7.10 by Ghaffari and Zuzic. To be precise, we need to compute the AncestorSum for each node where each virtual node inputs 1 and all other nodes input 0.

Note that the addition of virtual nodes preserves the graph's planarity, so each step can be executed in $\widetilde{O}(\text{HD})$ time, w.h.p. However, it remains to show that the value $\ell(v)$ we computed is in fact the correct level. For a block B_{i_j}, let $v_{i_j} \in \mathfrak{C}_i$ be the cut-node that is closest to root r_i (or is r_i itself). In the following, refer to each

[1] If not, we could also compute one in $\widetilde{O}(\text{HD})$ time, w.h.p.

edge that v_{i_j} replaced by a virtual node and a path as *marked*. Note that v_{i_j} is the only node that marks any edge in B_{i_j} as it is the only node with a parent outside of B_{i_j} or it is the unique root r_i. Further, by definition, the level of each cut-node in B_{i_j} is $\ell(v_{i_j}) + 1$ A simple induction over the levels shows that the level is equal to the number of marked edges to the root. For the induction beginning, consider all nodes that must assign themsleves level $\ell(v) = 0$. Clearly, the only node with level $\ell(v) = 0$ is the root r_i. This node trivially assigns itself level 0 in the algorithm. As all its incident edges are marked, no other node can assign itself level 0. For the induction step, assume that all nodes on some level $l \geq 0$ assign themselves the correct level. Recall that all cut-nodes in block a B_{i_j} except v_{i_j} get the same level. So w.l.o.g. we consider a block B_{i_j} and assume v_{i_j} has the correct level $\ell(v_{i_j}) = l$. It remains to show that each node in B_{i_j} has exactly one marked edge on its tree-path to v_{i_j}. This can be easily shown via contradiction: Recall that each node *must* be connected to v_{i_j} in T_i. If there were no marked edge, the path to v_{i_j} would use an edge in the parent-block. This is a contraction as the only path to the parent block is via v_{i_j}. Further, any path that contains one marked edge must have already reached v_{i_j}, so there cannot be another. Thus, there must be exactly one edge as claimed. Therefore, all node in B_{i_j} assign themselves level $\ell(v_{i_j}) + 1$. This proves that our implementation computes the correct value in $\tilde{O}(\mathrm{HD})$ time, w.h.p.

Step (S3) Equipped with the notion of levels, each node can locally check if it needs to add an A_i-edge. Recall that a cut-node v any adds an edge between two consecutive nodes w_i, w_{i+1}, if $\ell(\{v, w_i\}) = \ell(\{v, w_{i+1}\}) = \ell(v) + 1$. As v knows its own level and the level of all it edges, it can determine all its A-edges. Informing the neighboring nodes about the virtual edge can obviously be done in one round as both w_i and w_{i+1} are neighbors of v.

Step (S4) Let G_A be the graph that results from adding all A-edges. The embedding of each $\mathcal{C}_i^A$ can derrived from G_A's embedding. Thus, we will show that G_A has a planar embedding and that it can be computed in $\tilde{O}(\mathrm{HD})$ time. First, note that G_A it is a subgraph of $\widehat{G}$. By Lemma 9.11, $\widehat{G}$ is planar, and since planarity is closed under minor-taking, G_A is planar, too. Further, any τ-round CONGEST algorithm on $\widehat{G}$ (and therefore also G_A) can be computed in $O(\tau)$ rounds. Therefore, the embedding of G_A can be computed in time $\tilde{O}(\mathrm{HD})$ as claimed.

Step (S5) Knowing the planar embedding of G_A and all levels from the previous step is sufficient to compute the set B. Again, it is obvious that the neighboring nodes can be informed about the virtual edge in time $O(1)$.

□

Finally, Lemma 9.8 follows from Lemmas 9.9, 9.10, 9.11, 9.12, and 9.13. This concludes the analysis and this section.

9.2.4 Subroutine 3: Computing (Weighted) Path Separators

In this section, we prove a helpful intermediary result that might be of independent interest (although it was already remarked by Li and Parter in [LP19]). We consider Ghaffari and Parters' algorithm from [GP17] that constructs separators for biconnected components and show that it can be extended to node-weighted planar graphs. More precisely, we prove the following:

Lemma 9.14. *Let $G = (V, E, \omega)$ a node-weighted planar graph and let $\mathcal{C}_1, \dots, \mathcal{C}_N$ be a set of node-disjoint,* ***biconnected*** *subgraphs of G. Denote the weight of each component as*

$$w_i = \sum_{v \in V_i} \omega_v. \tag{9.5}$$

Further, let $T_1, \dots, T_N$ be series of spanning trees for $\mathcal{C}_1, \dots, \mathcal{C}_N$. Then, for any $\epsilon \in (0, 1/2)$, there is an algorithm that computes a separator S_i for each $(\mathcal{C}_i, T_i)$ such that each connected component in $\mathcal{C}_i \setminus S_i$ has weight at most

$$(1 + \epsilon) \cdot (2/3) \cdot w_i. \tag{9.6}$$

In particular, each separator consists of two paths in T_i and the two endpoints of an edge possibly not contained in $\mathcal{C}_i$. All separators can be computed in $\tilde{O}(\epsilon^{-2} \cdot HD)$ time, w.h.p.

Ghaffari and Parter already proved the corresponding lemma for unweighted graphs, i.e., graphs where all nodes have weight $\omega_v = 1$. The algorithm can easily be extended to work with weighted nodes. For the sake of completeness, we give a short overview of the required techniques. First, we note that the tree T_i for component $\mathcal{C}_i$ defines a dual-tree D_i which has a dual node v_F for every face F of $\mathcal{C}_i$ and an edge between two dual nodes if and only if their corresponding faces share a common non-T_i-edge. Such a tree **must** always exist. In the following, we refer to the nodes of T_i as *dual*-nodes and to the nodes of C_i simply as nodes. If we root the dual-tree D_i in an arbitrary dual-node v_{F_i}, this defines a subtree $D_i(v_F)$ for each dual-node v_F. We call the face obtained by merging all faces in the subtree of v_F its *superface* $\mathbb{S}_i(F)$. Note that the superfaces we consider are always defined with respect to the dual-tree D_i which in turn is defined by spanning tree T_i. Further, denote the set of nodes in each superface $\mathbb{S}_i(F)$ as $V[\mathbb{S}_i(F)]$. For any dual node v_F in dual-tree D_i, we define the weight of its superface (with respect to T_i) as:

$$w_{\mathbb{S}_i(F)} = \sum_{v \in V[\mathbb{S}_i(F)]} \omega_v$$

Consider an edge $\{v_F, v_{F'}\}$ in the dual-graph, then there must be an edge $\{v, w\}$ shared by faces F and F' that is not in T_i. Note that this is the **only** edge on the superface $\mathbb{S}_i(F)$ of v_F that is not part of T_i. Now consider the tree path P_v from the root to v and the tree path P_w from the root to w. Every path from a node within the superface to a node outside of the superface must cross this path. Thus, if we remove these two paths, all nodes within the face are separated from the remaining nodes. If we can find a superface $\mathbb{S}_i(F)$ of weight $w_{\mathbb{S}_i(F)}[w_i/(3(1+\epsilon)), 2(1+\epsilon)w_i/3]$, we can compute our desired separator using the two unique tree paths (and the additional non-tree edge) on its boundary.

Having defined all these concepts and intuitions, the algorithm works as follows: We assume that we have precomputed some arbitrary spanning tree T_i and all nodes know an approximation parameter $\epsilon \in (0, 1/2)$. First, we root the dual-tree D_i in some arbitrary dual-node v_F. Then, we compute the approximate weight

$\widehat{w}_{\mathbb{S}_i(F)}$ of each superface $\mathbb{S}_i(F)$ with respect to D_i. If there is a superface $\mathbb{S}_i(F)$ with approximate weight $\widehat{w}_{\mathbb{S}_i(F)} \in [\omega_i/(3(1+\epsilon)), 2(1+\epsilon)\omega_i/3]$, we choose the two T_i-paths that enclose the superface as separator. Otherwise, there must be a dual-node v_{F^*} whose superface $\mathbb{S}_i(F^*)$ has a weight more than $2(1+\epsilon)w_i/3$ and the superfaces of the children in D_i have weight smaller than $w_i/(3(1+\epsilon))$. In this case, we split the face F^* in two faces F^+ and F^- by adding a virtual edge on the face walk of F^*. Let $w_1, \dots, w_\delta$ be the nodes on the face walk of F^*. W.l.o.g. let $\{w_1, w_\delta\}$ be the edge that define the edge from v_{F^*} to its parent in D_i. Further, denote $w_{F_1}, \dots, w_{F_\delta}$ denote the children of v_{F^*} in clockwise order. Note that the edge to w_{F_j} in D_i is defined by edge $\{w_j, w_{j+1}\}$ in $\mathcal{C}_i$. Via binary search, we determine the child $w_{F_{j^*}}$ with $\sum_{j=1}^{j^*} \widehat{w}_{\mathcal{S}(F_{j^*})} \in [w_i/(3(1+\epsilon)), 2(1+\epsilon)w_i/3]$. Then, we **add a virtual edge that is not part of** G between w_1 and w_{j^*+1}. This splits face F^* into faces F^+ and F^-, s.t., the corresponding dual-node v_{F^-} is the new parent of $w_{F_1}, \dots, w_{F_{j^*}}$ while the other subface v_{F^+} is the parent of v_{F^-} and the remaining children. In particular, the superface $\mathbb{S}_i(F^-)$ has a weight within $[\omega_i/(3(1+\epsilon)), 2(1+\epsilon)\omega_i/3]$. Now, we return the two T_i-paths that enclose the corresponding superface of F^- as the separator.

For more details on the implementation, particularly the binary search to find $w_{F_{j^*}}$ in the last step, how the face is split, and how exactly the tree-path is chosen, we refer the interested reader to [GP17]. For our extension to weighted nodes, we consider all these steps correct and assume that they can be implemented in $\tilde{O}(\mathrm{HD})$ time, w.h.p. Figure 9.3 additionally summarizes the algorithm.

9.2.5 Analysis of ComputePathSep

We now begin with the analysis of the ComputePathSep. Note that our algorithm follows the algorithm of Ghaffari and Parter essentially verbatim. The only difference is that we compute the weights of the superfaces differently. We will slightly break our promise of being self-contained as we consider the following lemma to be true without presenting explicit proof.

Lemma 9.15 (Proven in [GP17]). *Suppose we have successfully identified a balanced or critical dual-node v_{F^*}. Then, we can compute a separator $\mathcal{S}_i$ that consists of two tree-paths in T_i and one edge $\{v, w\}$ that is not necessarily in $\mathcal{C}_i$ in $\tilde{O}(HD)$ time, w.h.p. Each component in $\mathcal{C}_i \setminus \mathcal{S}_i$ is of size at most*

$$(1+\epsilon) \cdot (2/3) \cdot w_i. \tag{9.7}$$

There is no direct lemma in [GP17] that makes this exact statement. However, it directly follows from the analysis and can verified with arguments provided in Sections B.2 and B.3.2 in the full version of [GP17]. The core idea for a balanced supernode is simply picking the the two tree paths on it boundary. For a critical supernode, one can find two nodes on the boundary, s.t. connecting them by an edge would create a balanced superface. In either case, the required operations are a series of minor aggregations.

Given this observation, the only step we need to touch for our extension to weighted nodes is Step (S3), where the algorithm decides whether there is a balanced or a critical dual-node. First, we need to show that the concepts of balanced and critical dual-node are well-defined with respect to any weight function. As it turns out, it is easy to verify that there must always be a balanced or critical supernode for any weight function ω. We do so in the following lemma:

```
ComputePathSep($\mathcal{C}'_i, T_i, \epsilon$):

(S1) Compute the dual-tree $D_i$ of $\mathcal{C}_i$ with respect to spanning tree $T_i$.
(S2) Root the dual-tree $D_i$ in an arbitrary face-node $v_{F_i}$. This defines a subtree $D_i(v_F)$ rooted in
     dual node $v_F$.
(S3) For each $v_F$, approximately compute the weight $\widehat{w}_F \in [(1-\epsilon)w_F, (1+\epsilon)w_F]$ of nodes in
     $D_i(v_F)$.
(S4) Determine, if there is a dual node $v_{F^*}$ with weight

         $\widehat{w}_F \in [w_i/(3(1+\epsilon)), 2(1+\epsilon)w_i/3]$

     If so, call it a balanced dual node. Otherwise, determine a dual node $v_{F^*}$ whose superface has
     at least weight $2(1+\epsilon)n/3$ but the superfaces of the children of $v_{F^*}$ have weight less than
     $n/(3(1+\epsilon))$ each. We call this a *critical* supernode.
(S5) We distinguish between two cases:
     (a) If there is a balanced supernode $v_{F^*}$, the $T_i$-path on the boundary of the superface $\mathbb{S}_i(F^*)$
         is the separator.
     (b) If there is a critical supernode $v_{F^*}$, add a virtual edge to create a face $F^-$ with
         $\widehat{w}_{F^-} \in \widehat{w}_F \in [w_i/(3(1+\epsilon)), 2(1+\epsilon)w_i/3]$. The $T_i$-path connecting the endpoints
         of this virtual edge is the separator.
```

Figure 9.3: Pseudocode for the ComputePathSep algorithm of Ghaffari and Parter [GP17] that computes a path separator for biconnected induced subgraph $\mathcal{C}_i$ given a spanning tree T_i.

Lemma 9.16. *Let D_i be the dual-tree for biconnected component $\mathcal{C}_i$ and a spanning tree T_i. For each weight function ω on the nodes V_i, there is a (not necessarily unique) balanced or critical dual-node v_{F^*} in D_i.*

Proof. For the proof, we will refer to the weight of a dual-node's superface with respect to D_i simply as its weight. As the algorithm computes a $(1 \pm \epsilon)$-approximation of the weight, it suffices to show that there is a superface with either weight $w_{F^*} \in [w_i/3, 2w_i/3]$ or its weight is larger than $2w_i/3$ and all its children weigh less than $w_i/3$. Now consider the following algorithm: Start at the root of dual-tree D_i and move the child with the largest weight until you reach a dual-node $v_{F^*} \in V[D_i]$ where all children have weight less than $w_i/3$ and return this dual-node. If two children have the same weight, break ties arbitrarily but consistently. Note that the algorithm always returns a node as, eventually, we reach a leaf dual-node where the condition is trivially fulfilled because it has no children. Thus, v_{F^*} is a well-defined dual-node. By construction, the v_{F^*}'s weight must be greater than $\omega_i/3$. Otherwise, we would not have moved to v_{F^*} from its parent. Further, all its children have a weight less than $w_i/3$. Otherwise, we would not have stopped to v_{F^*} and returned it. Therefore, v_{F^*} must be critical or balanced. This proves the lemma. $\square$

Thus, it remains to approximately count the size of all superfaces. Recall that all nodes that are part of face/dual node v_F form a connected component in the face graph $\mathcal{F}(\mathcal{C}_i)$. Further, two neighboring dual nodes v_F and $v_{F'}$ share an edge $\{v, w\} \in E_i$, and we can merge all neighboring faces without violating planarity by adding extra edges in the face graph. Thus, we can perform one round of the minor aggregation model from Theorem 7.1.1 in $\tilde{O}(\mathrm{HD})$ time, w.h.p. In particular, we can perform the tree operations from Lemma 7.10 time, w.h.p. This allows us to compute an aggregate function on all faces of superface $\mathbb{S}_i(F)$ in $\tilde{O}(\mathrm{HD})$ time, as the corresponding dual-nodes are descendants in the dual-tree D_i. However, recall that each node $v \in V$ is part of up to $\mathsf{deg}(v)$ faces, so we must be careful not to overcount. To this end, Ghaffari and Parter present a randomized sketching technique, which ensures that a node is (approximately) counted only once. For details on this technique, we refer to [GP17]. We present a slightly different technique to compute the weights that can be easier combined with our established techniques. In particular, we use the generic approximation result from [MS06]. On a high level, it states that if we can aggregate the minimum of a set of exponentially distributed random variables, we can efficiently approximate any sum of integers. More precisely, it holds:

Lemma 9.17 (Approximation through Aggregation, cf. [MS06]). *Let $W = \{v_1, \dots, v_{|W|}\}$ be a subset of nodes where each node $v_i \in W$ has a positive weight $\omega_i \in \mathbb{N}$. Define $w_W = \sum_{i=1}^{|W|} \omega_i$ as the sum of all weights. For each $v_i \in W$, define r independent, identically distributed random variables $X_i^1, \dots, X_i^r$ that are exponentially distributed with parameter ω_i. Formally, for all $j \in [1, r]$, it holds $X_i^j \sim \mathsf{Exp}(\omega_i)$. Consider the value:*

$$\widehat{w}_W^{(r)} = \frac{r}{\sum_{j=1}^{r} X_W^j}$$
$$\textit{s.t.} \quad X_W^j := \min\left\{X_1^j, \dots, X_{|W|}^j\right\} \quad \forall j \in [1, r]$$

Then, it holds for $r \geq 1$ and $\epsilon \in (0, 1/2)$ that

$$\mathbf{Pr}\left[\left|w_W - \widehat{w}_W^{(r)}\right| \geq 2\epsilon w_W\right] \leq 2e^{-\frac{\epsilon^2 \cdot r}{3}}$$

To put it more simply, the lemma implies that $r \in O(\epsilon^{-2}\log n)$ minor aggregations are sufficient to approximate the weight of any subset of nodes. Note that we can only aggregate an approximate minimum of the exponential random variables' outcomes. Since our minor aggregations are limited to $O(\log n)$ bits and the exponential distribution is continuous, we must cut off the variables at a certain point and only send the $O(\log n)$ most significant bits. However, the difference between the actual value and the value we send is within a magnitude of $o(n^{-c})$ for an arbitrary constant c. Thus, if we consider a large enough $\epsilon \in \omega(n^{-c})$, this additional imprecision does not affect the result meaningfully. Equipped with this insight, we can prove the following:

Lemma 9.18. *Let $D_1, \dots, D_N$ be a collection of dual-trees of node-disjoint components $\mathcal{C}_1, \dots, \mathcal{C}_N$ of a node-weighted planar graph $G = (V, E, \omega)$. Then for each superface $\mathbb{S}(F)$ in any of the trees, we can compute its approximate weight $\widehat{w}_{\mathbb{S}(F)} \in [(1-\epsilon)\widehat{w}_{\mathbb{S}(F)}, (1+\epsilon)\widehat{w}_{\mathbb{S}(F)}]$ in $\tilde{O}(HD)$ time, w.h.p.*

Proof. We want use Lemma 9.17 to compute the approximate values. To this end, the computation proceeds in $r \in O(\epsilon^{-2}\log n)$ iterations. For a component $\mathcal{C}_i$, a single iteration $j \in [0, r]$ works as follows:

(S1) First, each node $v \in V_i$ with positive weight ω_v draws an exponentially distributed random variable X_v^j with parameter ω_v. We denote the realization of that random variable as x_v^j, and we define $\widehat{x}_v^j$ to be the value of x_v^j truncated to the first $c \log n$ most significant bits. It clearly holds:

$$|x_v^j - \widehat{x}_v^j| \leq \frac{1}{n^c}$$

(S2) Then, we construct the face graph $\mathcal{F}(\mathcal{C}_i)$. Each node $v \in V_i$ passes its value $\widehat{x}_v^j$ to all its (virtual) face nodes in $\mathcal{F}_i$. Then, in each face F, we aggregate the minimal value of all $\widehat{x}_v^j$ that belong to that face. For each face F let $\widehat{x}_F^j$ be the corresponding aggregated value.

(S3) Then, in each dual-tree D_i, we compute SubsetSum where each dual-node v_F chooses $\widehat{x}_F^j$ as input value and we aggregate the minimum of inputs. Therefore, each v_F learns the minimum of all $\widehat{x}_{F'}^j$, of all faces $F' \in \mathbb{S}_i(F)$ in its superface $\mathbb{S}_i(F)$. We denote the resulting value as $\widehat{x}_{\mathbb{S}_i(F)}^j$.

After all r iterations are finished, each dual-node v_F has learned r values $\widehat{x}_{\mathbb{S}_i(F)}^1, \dots, \widehat{x}_{\mathbb{S}_i(F)}^r$. Using these values, each dual-node v_F computes the following value:

$$\widehat{w}_{\mathbb{S}_i(F)} = \frac{r}{\sum_{j=1}^r \widehat{x}_{\mathbb{S}_i(F)}^r}$$

We will show that $\widehat{w}_F$ is an $(1 \pm \epsilon)$-approximation of $\widehat{w}_F$. As computation $\widehat{w}_F$ resembles the formula given in Lemma 9.17, it suffices to show that for each $j \in [0, r]$, it holds:

$$\widehat{w}_{\mathbb{S}_i(F)}^j := \min \left\{ \widehat{x}_v^j \mid v \in V[\mathbb{S}_i(F)] \right\}$$

However, this follows directly from the computation given above. Note that by definition, it holds:

$$\widehat{w}_{\mathbb{S}_i(F)}^j := \min \left\{ \widehat{x}_{F'}^j \mid F' \in \mathbb{S}_i(F) \right\}$$

Further, for each face $F' \in \mathbb{S}_i(F)$ and each iteration $j \in [0, r]$, it holds by construction:

$$\widehat{x}_{F'}^j := \min \{ \widehat{x}_v^j \mid v \in W_{F'} \}$$

Thus, by combining these two facts, we get the desired result.

Given the correctness of the approximation, it remains to prove that we can implement this in $\tilde{O}(\epsilon^{-2} \cdot \mathrm{HD})$ time, w.h.p. Each iteration consists of one minor aggregation in each face graph $\mathcal{F}(\mathcal{C}_\rangle)$ and $\tilde{O}(1)$ minor aggregation in each dual-tree D_i. In Step (S2), we need to aggregate the minmum of all nodes of the same face F. All faces are node-disjoint connected components in $\mathcal{F}(\mathcal{C}_i)$, so this can be done with a single minor aggregation. Therefore, it can done on all faces in $\tilde{O}(\epsilon^{-2} \cdot \mathrm{HD})$ time, w.h.p. In Step (S3), we only perform SubsetSum on each tree $D_1, \dots, D_N$. SubsetSum requires $\tilde{O}(1)$ minor aggregations and therefore can be done in $\tilde{O}(\mathrm{HD})$ time, w.h.p., on all trees. Given that there are $O(\epsilon^{-2} \log n)$ iterations, all values can be determined in $\tilde{O}(\epsilon^{-2} \cdot \mathrm{HD})$ time, w.h.p. □

Given these approximate weights, any balanced dual-node can immediately identify itself. We then aggregate the maximal identifier of all balanced dual-nodes in $\mathcal{C}_i$ and continue the algorithm with the resulting dual-node. Aggregating all the maximal identifiers in all components can be done $\tilde{O}(\mathrm{HD})$ time, w.h.p.

Otherwise, if there is no balanced dual-node, we need to do a bit more work to find a critical supernode. In the following, we denote all dual-nodes v_F with $w_{\mathbb{S}_i(F)} \leq w_i/(3(1+\epsilon))$ as *light* and all dual-nodes with $w_{\mathbb{S}_i(F)} \geq 2(1+\epsilon)w_i/3]$ as *heavy*. Note that there cannot be dual-nodes in between these weight bounds as they would be balanced. We then let each dual-node aggregate the number of its light children. This can be done is $\tilde{O}(\mathrm{HD})$ time, w.h.p. Any *heavy* dual-node v_F where *all* its children are light, marks itself as a critical dual-node. Again, we aggregate the maximal identifier if there is more than one. Thus, together with computation of the weights, we can identify a balanced or critical dual-node in $\tilde{O}(\epsilon^{-2} \cdot \mathrm{HD})$ time, w.h.p. Together with Lemma 9.15, this implies Lemma 9.14 and concludes this section.

9.2.6 Main Algorithm Description & Analysis (Proof of Lemma 9.1)

In this section, we combine all subroutines from the previous three sections and finally present and analyze the full algorithm behind Lemma 9.1. Recall that this algorithm must create path separators in *arbitrary* connected node-disjoint subgraphs $\mathcal{C}_1, \dots, \mathcal{C}_N$ of an arbitrary connected planar graph G. Following the sketch in the beginning, the algorithm proceeds in four synchronized steps:

(Step 1) Creates Buffers between Components: Separate all components by adding (virtual) nodes on their outgoing edges.

1. For each outgoing edge $\{v, w\} \in E$ to a node $w \notin \mathcal{C}_i$, we add the virtual nodes $w_i \in V_i'$. Note that the component $\mathcal{C}_j$ that contains w may also add a virtual node v_j.
2. Replace the edge $\{v, w\}$ by a path from v to w via the virtual nodes. In the resulting graph, all components are well-separated.
3. Mark the edges between the real nodes and the virtual nodes to the respective spanning tree T_i, thereby obtaining a tree T_i'.

Denote the graph that results from adding the virtual nodes as G'. We will work on G' instead of G for the rest of the algorithm.

(Step 2) Make Components Biconnected: Execute MakeBiconnected (cf. Figure 9.2) on all components $\mathcal{C}_1, \dots, \mathcal{C}_N$. Denote the resulting biconnected components as $\mathcal{C}_1', \dots, \mathcal{C}_N'$. Note that all $\mathcal{C}_1', \dots, \mathcal{C}_N'$ are node-disjoint **biconnected** supersets of $\mathcal{C}_1, \dots, \mathcal{C}_N$.

Recall that MakeBiconnected adds virtual edges to G' and denote the graph results from adding these virtual edges as G''. We will work on G'' instead of G' for the rest of the algorithm.

(Step 3) Compute Separators: Assign each virtual node $v \in V' \setminus V$ weight $\omega_v = 0$ and each real nodes $w \in V$ weight $\omega_w = 1$. Them, execute ComputePathSep (cf. Figure 9.3) on all components $\mathcal{C}'_1, \dots, \mathcal{C}'_N$ with their respective trees $T'_1, \dots, T'_N$ and approximation parameter $\epsilon = 1/8$. This produces a collection of path separators $\mathcal{S}'_1, \dots, \mathcal{S}'_N$ for $\mathcal{C}'_1, \dots, \mathcal{C}'_N$.

(Step 4) Prune Virtual Nodes: If any separator $\mathcal{S}'_i$ for component $\mathcal{C}'_i$ contains any virtual node $v \in V'$, remove it. Return the resulting separator $\mathcal{S}_i := \mathcal{S}'_i \cap V_i$ as separators for $\mathcal{C}_i$.

Analysis

The correctness and runtime of the resulting algorithm follow straightforwardly from the subroutines we have proven in the preceding sections. For the sake of completeness, we quickly go through all four steps. The first step trivially ensures that the components $\mathcal{C}_1, \dots, \mathcal{C}_N$ are well-separated. Given a well-separated components, MakeBiconnected returns biconnected supersets $\mathcal{C}'_1, \dots, \mathcal{C}'_N$ as per Lemma 9.8. Thus, the sets $\mathcal{C}'_1, \dots, \mathcal{C}'_N$ (along with their spanning trees) are valid inputs for ComputePathSep. Note that, for our weight fuction ω, the total weight of ω_i of components $\mathcal{C}'_i$ is equal to $|V_i|$, the number of nodes in $\mathcal{C}_i$. For a parameter $\epsilon := 1/8$, the algorithm ComputePathSep$(\mathcal{C}'_i, T'_i, \epsilon)$ creates separators such that the biggest connected component in each $\mathcal{C}'_i \setminus \mathcal{S}'_i$ is of size at most

$$(1+\epsilon)\cdot(2/3)\cdot|V_i| \leq (1+1/8)\cdot(2/3)\cdot|V_i| \leq (12/24+2/24)\cdot|V_i| \leq (16/24)\cdot|V_i| \leq (3/4)\cdot|V_i|. \quad (9.8)$$

This follows from Lemma 9.14. Finally, we must argue that removing the potential virtual nodes from a separator $\mathcal{S}'_i$ must turn it into a separator for $\mathcal{C}_i$ that consists of tree paths in T_i. Formally, we show that it holds:

Lemma 9.19. *Let $\mathcal{S}'_i$ be a computed separator for $\mathcal{C}''_i$ and let $\mathcal{S}_i$ be the separator that results from removing all virtual nodes from $\mathcal{S}'_i$. Then, $\mathcal{S}_i$ is a separator for $\mathcal{C}_i$ and consists of two paths from T_i and an edge $\{v, w\}$ that may not be part of G.*

Proof. We prove the two statements separately. For the first statement, we need to show that each connected component in $\mathcal{C}_i \setminus (\mathcal{S}'_i \cap V'_i)$ has a weight of at most $(1+\epsilon)2\omega_i/3$. We prove this by gradually stripping away the additional (virtual) nodes and edges we added in our auxiliary procedures. To this end, we consider the two auxilliary graphs $\overline{\mathcal{C}}'_i$ and $\overline{\mathcal{C}}_i$ defined as follows:

$$\overline{\mathcal{C}}'_i = \mathcal{C}''_i \setminus (\mathcal{S}'_i \cup V'_i)$$
$$\overline{\mathcal{C}}_i = \overline{\mathcal{C}}'_i \setminus (A_i \cup B_i)$$

In other words, $\overline{\mathcal{C}}'_i$ is the graph that results from removing the separator $\mathcal{S}'_i$ and all virtual nodes from V'_i from $\mathcal{C}''_i$. Further, $\overline{\mathcal{C}}_i$ is obtained from $\overline{\mathcal{C}}'_i$ by removing the remaining virtual edges added by MakeBiconnected$(\mathcal{C}_i)$. Note that $\overline{\mathcal{C}}_i$ induced subgraph of all nodes $v \in V_i$ in G that are not part of $\mathcal{S}'_i$. Thus, it is equivalent to $\mathcal{C}_i \setminus (\mathcal{S}'_i \cap V'_i)$. Given this insight, recall that each connected component in $\mathcal{C}''_i \setminus (\mathcal{S}'_i)$ a weight of at most $(1+\epsilon)2\omega_i/3$ by Lemma 9.14. Thus, it suffices to show that the heaviest connected components of $\mathcal{C}''_i \setminus \mathcal{S}'_i$ is heavier than the heaviest component of $\overline{\mathcal{C}}_i$. To get from $\mathcal{C}_i'' \setminus \mathcal{S}_i'$ to $\overline{\mathcal{C}}_i$, we only remove more nodes and edges. Thus, the resulting components can only be smaller and lighter. This is proves that $\mathcal{S}_i = \mathcal{S}_i' \cap V_i$ is a separator for $\mathcal{C}_i$.

For the second property, we consider the spanning T_i' obtained from T_i by adding the edges to the virtual nodes. Recall that the separator $\mathcal{S}_i'$ only contains paths from T_i' and a (possibly) non-existent edge between the endpoints of these paths. We added the virtual edges **after** we have computed T_i'. By this construction, $\mathcal{S}_i$ will not contain a tree path where two nodes are connected via (virtual) edge from A_i or B_i. However, the additional edge might be between two virtual nodes using a virtual edge and the paths may contain one or more virtual nodes. First, consider the tree paths. By closely observing the construction, we see that the virtual nodes must be leaves in the tree T_i'. Now suppose that the separator contains a path P in T_i that contains a virtual node $v \in V_i$. As v is a leaf, it *must* be the last node on the path. Thus, if we remove it, we obtain a proper path in T_i that only consists of real nodes from V_i. Second, consider the additional edge $\{v, w\}$. If this edge is between two real nodes $v, w \in V$, we can keep it as is. Otherwise, if either endpoint is virtual, we must find another suitable edge. W.l.o.g. assume both v and w are virtual and let v' and w' the respective real nodes that simulate them. If we contract both $\{v', v\}$ and $\{w', w\}$, this creates an edge $\{v', w'\}$ (if it not existed already). We can take that edge (which does not necessarily exist in $\mathcal{C}_i$) instead. This proves the lemma. □

Next, we will show that the algorithm can be efficiently implemented in $\tilde{O}(\mathrm{HD})$ time, w.h.p. It holds:

Lemma 9.20. *The algorithm can be implemented in $\tilde{O}(HD)$ time, w.h.p.*

Proof. As an auxiliary result, we first show that adding virtual nodes does not violate planarity and the hop diameter of G' is at most $3 \cdot \mathrm{HD}$. To this end, note that we can place an arbitrary number of nodes on an edge without violating the planarity of the resulting graph. It is easy to see that this maintains planarity when considering a drawing of G: If we draw the additional nodes on each line that represents an edge, we obtain a drawing of G' where no lines cross. Thus, G' is planar. Further, the hop diameter of G' is at most $3 \cdot \mathrm{HD}$ as we add at most two virtual nodes on each edge.

Further, we can simulate τ-round CONGEST algorithms on G' in τ rounds on G. To do this, all real nodes will simulate their adjacent virtual nodes. Consider an edge $\{v, w\}$ between components and the virtual nodes w' and v' added by v and w, respectively. Further, we add the virtual edges $\{v, w'\}$, $\{v', w'\}$, and $\{v', w\}$. Each round of communication in graph G' can be simulated as one round of communication on G. Each node v that simulates w' simply sends the corresponding messages to w, which simulates v'. This only increases the amount of messages sent from v to w by a constant factor.

With these preliminaries out of the way, we now prove the algorithm's runtime step by step. In the first step, all real nodes exchange the identifiers of their simulated virtual with their neighbors. This is required for the simulation and takes one round. In the second step, we simulate MakeBiconnected on G'. Adding virtual nodes causes all components $\mathcal{C}_1, \dots, \mathcal{C}_N$ to be well-separated. Given that the graph G' is planar and has diameter $3 \cdot \mathrm{HD}$, the runtime of MakeBiconnected is $\tilde{O}(\mathrm{HD})$, w.h.p. The resulting graph G'' is still planar, and we can simulate any CONGEST algorithm with constant overhead. All of this follows from Lemma 9.8. Thus, by Lemma 9.14, we can execute ComputePathSep in $\tilde{O}(\mathrm{HD})$ time, w.h.p., as well. Finally, removing the virtual nodes is a local operation. This proves the lemma. □

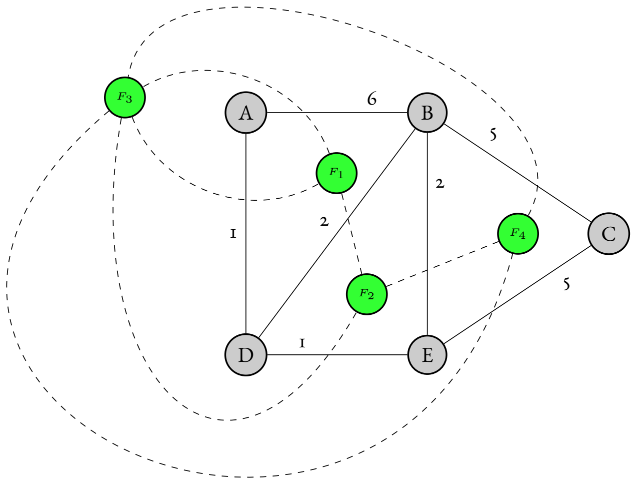


Figure 9.4: The dual graph of a planar graph G. Each green node F_1, F_2, F_3, F_4 represents a face. The dotted lines are edges between faces. Any algorithm on the face graph can simulated in G with constant factor blowup.

Thus, by Lemma 9.20 our algorithm can be implemented in $\tilde{O}(\mathrm{HD})$ time, w.h.p., and by Lemma 9.19 it produces the desired separators for each subgraph. This concludes the analysis of Lemma 9.1 and this section.

9.3 Fast Separators in the PRAM and the HYBRID Model

In this section, we will prove (the HYBRID part of) Lemma 9.1. Due to the additional communication capabilities of the HYBRID model, this will be a much easier task than for CONGEST. For an easier explaination of the main ideas, we will first consider the problem of computing a cycle separator for a planar graph G given a spanning tree T in the PRAM. Then, we will argue how to translate the algorithm to the HYBRID model and how to execute it in node-disjoint subgraphs in parallel.

As our main tool, we employ the algorithm of Kao, Teng, and Toyama presented in [KTT93]. This algorithm finds a separator in an undirected planar graph in depth $O(\log n)$ for the PRAM model. To be precise, Kao, Teng, and Toyama show the following:

Lemma 9.21 (Implied by Theorem 3.1 in [KTT93]). *Let $G = (V, E)$ be a planar graph with n nodes. Further, let π be an embedding of G and T be a spanning tree for G. Then, there is an algorithm that finds a path P in T, such that each connected component in $G \setminus P$ has at most $(2/3) \cdot n$ nodes. Path P can be computed in $O(\log n)$ depth with and $O(n)$ work on $\frac{n}{\log n}$ processors in the PRAM.*

Note that the precise statement of the theorem in [KTT93] is different from that presented here. However, our revised statement reflects more precisely what their algorithm is doing internally. Now, given a path P

with the properties mentioned above, it is easy to derive a cycle separator we require for Lemma 9.1. Supposing that path P goes from nodes v to w, return the unique paths in T from s to v and w. This is the desired cycle separator for G as it is a superset of P. We can compute a (combinatorial) planar embedding π of G in PRAM using the algorithm of [RR89]. This assigns a clockwise order to all edges in G. The algorithm requires $O(\log n)$ depth and linear work. Putting these results together provides us with an algorithm that, given a tree T, computes a cycle separator in logarithmic depth and linear work in the PRAM.

Next, we show how this algorithm can be brought to the HYBRID model. As a planar graph has constant arboricity (because it excludes K_5 as a minor), simulating this PRAM algorithm by using the framework of [FHS20] immediately yields a HYBRID algorithm with runtime $O(\log^2 n)$ w.h.p. For details on the simulation, see the proof of Lemma 7.4 in Chapter 7. More precisely, we can simultaneously simulate the algorithm in each connected component $\mathcal{C}_1, \ldots, \mathcal{C}_N$ and construct a separator for all them in $O(\log^2 n)$ time, w.h.p. This proves the HYBRID part of Theorem 7.

Harder, better, faster, stronger.

DaftPunk

10 Strong Low-Diameter Decompositions Via Approximate Distances

In this chapter, we consider the efficient construction of so-called (probabilistic) low-diameter decompositions (LDD) for general and restricted graphs in the CONGEST and the HYBRID model. All algorithms and techniques that are presented in this chapter have been published in the following paper:

> Jinfeng Dou, Thorsten Götte, Henning Hillebrandt, Christian Scheideler, and Julian Werthmann. Distributed and parallel low-diameter decompositions for arbitrary and restricted graphs. In Raghu Meka, editor, *16th Innovations in Theoretical Computer Science Conference (ITCS 2025), New York City, NY, USA, January 7-10, 2025*, Leibniz International Proceedings in Informatics (LIPIcs), page (to appear), Dagstuhl, Germany, 2025. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik

An LDD of a (weighted) graph $G := (V, E, \ell)$ is a partition of G into disjoint subgraphs $\mathcal{K}(G) := K_1, \dots, K_N$ with $K_i := (V_i, E_i)$. We will refer to these subgraphs as *clusters*. In a partition, each node is contained in exactly one of these clusters. An algorithm that constructs an LDD is parameterized with a value $\mathcal{D} > 0$, which gives an upper bound on the clusters' diameter. We say that a clustering has *strong* diameter $\mathcal{D}$ if each K_i has a diameter of at most $\mathcal{D}$. That is, between two nodes in K_i there is a path of length (at most) $\mathcal{D}$ that only consists of nodes in K_i. In contrast, an LDD with a weak diameter creates possibly disconnected subgraphs with a diameter of $\mathcal{D}$ with respect to the original graph G. Here, for all nodes $v, w \in K_i$, there is a path of length (at most) $\mathcal{D}$ from v to w that can use nodes outside of K_i. We measure the decomposition quality by the number of edges that are *cut* between clusters. An edge $\{v, w\} \in E$ is cut if its two endpoints v and w are assigned to different clusters. Thus, a good clustering algorithm only ensures that *most* nodes are in the same cluster as their neighbors. Without this constraint, a trivial algorithm could simply remove all edges from the graph and return clusters containing one node each.

There are several ways to count the edges that are cut. In this thesis, we consider so-called *probabilistic* decompositions. Here, the probability for an edge $z = \{v, w\}$ to be cut between two clusters K_i and K_j with $i \neq j$ that contain its respective endpoint depends on its length ℓ_z. This means that *short* edges are cut less likely than *long* edges. We use the following formal definition of LDD's in the remainder of this work:

Definition 10.1 (Probabilistic Low-diameter Decomposition (LDD)). *Let $G := (V, E, \ell)$ be a weighted graph and $\mathcal{D} > 0$ be a distance parameter. Then, an algorithm computes a LDD with quality $\alpha \geq 1$ creates a series of disjoint clusters $\mathcal{K} := K_1, K_2, \ldots, K_N$ with $K_i := (V_i, E_i)$ where $V_1 \sqcup \ldots \sqcup V_N = V$. Futher, it holds:*

1. ***Strong Diameter:** Each cluster $K_i \in \mathcal{K}$ has a strong diameter of at most $\mathcal{D}$.*
2. ***Low Edge-Cutting Probability:** Each edge $z := \{v, w\} \in E$ of length ℓ_z is cut between clusters with probability (at most) $O\left(\frac{\ell_z \cdot \alpha}{\mathcal{D}}\right)$.*

If we do not require each node to be in a cluster, we use the related notion of a Low-Diameter *Clustering* (LDC). In a LDC with quality (α, β) and strong diameter $\mathcal{D}$ we create a series of disjoint clusters $\mathcal{K} := K_1, K_2, \ldots, K_N$ with $K_i := (V_i, E_i)$. Each cluster $K_i \in \mathcal{K}$ has a strong diameter of at most $\mathcal{D}$ and each edge $z := \{v, w\} \in E$ of length ℓ_z is cut between clusters with probability (at most) $O\left(\frac{\ell_z \cdot \alpha}{\mathcal{D}}\right)$. Further, each node is part of a cluster with probability at least β. As we will see, LDD's and LDC's are nearly equivalent as we can build an LDD from an LDC by repeatedly applying it until all nodes are clustered.

Having defined them, we can discuss the natural question why we need LDD's in the first place. Simply put, LDD's are part of an algorithm designer's toolkit for developing efficient divide-and-conquer algorithms, as they offer a generic way to decompose a given graph. As such, creating LDD's with low edge-cutting probability plays a significant role in numerous algorithmic applications. In the following, we present a comprehensive list of applications (which we do not claim to be complete):

Tree Embeddings: For example, they can be used to embed graphs into trees, i.e., constructing so-called metric tree embeddings (MTE) and low-stretch spanning trees (LSSP). In both, we map a given graph $G := (V, E, \ell)$ into a randomly sampled tree $T = (V, E_T, \ell_T)$ in a way that preserves the path lengths of the original graph G on expectation. In an LSSP, it must hold $E_T \subset E$, i.e., we compute a subtree of G, while a MTE can use virtual edges not present in the original graph. LSSPs and MTEs have proven to be a helpful design paradigm for efficient algorithms as many computational problems are significantly easier to solve on trees than on general graphs. For example, they have been used to construct distributed and parallel solvers for certain important classes of LPs [BGK$^+$11, ALH$^+$23, dV23, GKK$^+$18], which can then be used solve MaxFlow and other complex problems. Nearly all constructions use LDD's (or variants thereof) as subroutines; see [ABN08, AN19, BEGL19, EEST08] for the construction of LSSPs and [Bar96, Bar98, Bar04, FRT04] for MTEs.

Light Spanners: Another interesting avenue for LDD's are so-called *light spanners* first introduced by Elkin, Neimann, and Solomon in [ENS15]. Given a weighted graph $G = (V, E, \ell)$, a t-spanner is a subgraph $H \subset G$

that approximately preserves the distances between the nodes of the original graph by a factor t, i.e., for all pairs of node $v, w \in V$, it holds $d_H(v,w) \leq t \cdot d_G(v,w)$. A spanner's lightness L_H is the ratio between the weight of all its edges and the MST of graph G, i.e., it holds:

$$L_H = \frac{\sum_{e \in H} \ell_e}{\sum_{e \in MST(G)} \ell_e}$$

Thus, compared to the LSSPs and MTEs, light spanners have more freedom in choosing their edges as they are not required to be trees. The lightness is (arguably) a natural quality measure for a spanner, especially for distributed computing. Consider, for example, an efficient broadcast scheme in which a node wants to spread a message to every other node in G. Further, suppose the edges' length as the cost of using this edge to transmit a message. If we send the messages along the edges of the light spanner H of G, the total communication cost can be bound by its lightness, and the stretch bounds the cost of the path between any two nodes. Despite this, there are almost no distributed algorithms that t-spanners with an upper bound on the lightness with [EFN20] being a notable exception of stretch $O(k)$ with lightness $\tilde{O}(n^{\frac{1}{k}})$. The complicating factor in the distributed construction is that lightness is a global measure. Many previously known distributed spanner constructions like [BS07] or [MPVX15a] are very local in their computations as they only consider each node's t-neighborhood with $t \in O(\log n)$. Therefore, the lightness of the resulting spanners is unbounded despite them having few edges.

LDD's are connected to light spanners through an observation by Neiman and Filtser in [FN22]: They show that with black-box access to an LDD algorithm with quality α, one can construct an $O(\alpha)$-spanner with lightness $\tilde{O}(\alpha)$ for any weighted graph that $G = (V, E, \ell)$. In addition to an algorithm that creates LDD's for geometrically increasing diameters, they only require so-called (α, β)-nets for which they already provided a distributed implementation in [EFN20]. Thus, finding better distributed algorithms LDDs, especially for restricted graphs, drectly improves the distributed construction of light spanners.

Compact Routing Schemes: *Routing schemes* are distributed algorithms that manage the forwarding of data packets between network devices. Thus, developing efficient routing schemes is essential to enhance communication among multiple parties in distributed systems. It is known that LDD's can be used to construct routing schemes with routing paths that are not significantly longer than the shortest paths in G. In particular, an LDD with strong diameter and quality α directly enables us to construct a routing scheme where all paths only differ by a factor of $O(\alpha)$ from the true shortest paths. We elaborate in this in Chapter 11, which is dedicated to finding such routing schemes. There, we present two efficient construction of routing schemes with the help of LDD's.

Despite this vast number of applications for LDD's on *weighted* graphs, research on LDD's in a distributed setting has mostly focused on the unweighted case, producing many efficient algorithms in this regime. In particular, the research focussed on so-called *network decompositions* that enable fast algorithms for local problems like MIS, Coloring, or Matching. In principle, these algorithms could be applied to weighted graphs. However, their runtime depends on the weighted diameter $\mathcal{D}$ of the resulting clusters, which might be much larger than the hop diameter (or even n). We give a more detailed overview of these algorithms in Section 10.6. Two notable exceptions explicitly consider weighted graphs and are closely related to our results: The work of Becker, Emek, and Lenzen [BEL20] creates LDD's of quality $O(\log n)$ with weak diameter[1] for general graphs. We note that $O(\log n)$ is the best quality we can hope for in an LDD due to a result by Bartal[Bar96]. The algorithm requires $\tilde{O}(\mathrm{HD} + \sqrt{n})$ in the CONGEST model, which is optimal as each distributed LDD construction in a weighted graph requires $\Omega(\mathrm{HD} + \sqrt{n})$ time [GZ22b] as we can derive approximate shortest paths from it. Just as our algorithm, [BEL20] consists of $\tilde{O}(1)$ $(1+\epsilon)$-approximate SetSSP computations with $\epsilon \in O\left(1/\log^2 n\right)$. Further, there is the work of Rozhon, Elkin, Grunau, ⓡ Haeupler [REGH22], which makes two significant improvements compared to [BEL20]. They present a decomposition with strong diameter (instead of weak), and their construction is deterministic (instead of randomized). Conversely, they have a slightly worse quality of only $O(\log^3 n)$. Again, the algorithm consists of $\tilde{O}(1)$ $(1+\epsilon)$-approximate SetSSP computations with $\epsilon \in O\left(1/\log^2 n\right)$. For restricted graphs like planar, bounded treewidth, or minor-free graphs, we are unaware of a distributed algorithm explicitly designed for weighted graphs.

In this chapter, we further deepen the understanding of LDD's in weighted graphs by (in certain ways) improving upon the algorithms for both general and restricted graphs. The following theorem summarizes the main results of this chapter:

Theorem 8. *A Clustering Theorem for Restricted Graphs*
Let $\mathcal{D} > 0$ be an arbitrary distance parameter and $G := (V, E, \ell)$ be a (possibly weighted) undirected graph. Then, the following two statements hold:

- For arbitrary graph G, there is an algorithm that creates an LDD of G with strong diameter $\mathcal{D}$ and quality $O(\log n)$.
- If G is $\tilde{O}(1)$-path separable, then there is an algorithm that creates an LDD of G with strong diameter $\mathcal{D}$ and quality $O(\log\log n)$.

The algorithms can be implemented in $\tilde{O}(1)$ minor aggregations and $\tilde{O}(1)$ $(1+\epsilon)$-approximate SetSSP computations with $\epsilon \in O\left(1/\log^2 n\right)$.

By Lemma 7.9, this implies that the algorithm can, w.h.p., be implemented in $\tilde{O}(\mathrm{HD} + \sqrt{n})$ time in CONGEST and $\tilde{O}(1)$ time in HYBRID with local capacity of $\lambda \in O(\log n)$ and a global capacity of $O(\log^2 n)$. Before we go into the details of the promised algorithms, let us discuss the implications of this theorem. First, we note that our algorithm is currently the best randomized LDD construction in CONGEST for general weighted graphs. It has same runtime as [REGH22] and [BEL20], creates clusters of *strong* diameter, and has (asymptotically) optimal quality of $O(\log n)$. Thus, compared to [REGH22], it is faster (but randomized), and compared to [BEL20], it creates clusters of strong diameter. For k-path separable graphs, the situation is more nuanced. Recall that each universally k-path separable graph excludes K_{4k+1} as minor, and thus, we need to compare ourselves to algorithms for K_r-free graphs. As mentioned above, there are distributed algorithms [LMR21, CS22] for unweighted graphs that present LDD's in restricted graphs with quality $O(r)$. While our

[1] Actually, [BEL20] proves a stronger property of the diameter. Although the diameter is weak, the number of nodes outside of a cluster that are part of the shortest path between two nodes of a cluster is limited. For many applications, this is sufficient and just as good as a strong diameter.

bound of $O(\log \log n)$ is exponentially better than that for general graphs, it is still far from these bounds as it depends on n and not only on r. On the flip side, our algorithm has an (asymptotically) optimal runtime of $\tilde{O}(\mathrm{HD})$ where the hidden factors depend on r. The other distributed algorithms are tailored to small clusters, as the runtime is polynomial in the cluster's diameter $\mathcal{D}$. Therefore, our algorithm is faster for large diameters $\mathcal{D}$, which arguably trades off its worse cutting probability.

10.1 Structure of this Chapter

The *roadmap* for this chapter is as follows. We begin with three technical sections that introduce some useful tools and auxiliary algorithms that we will use throughout this chapter (and will also reuse in the next). In particular, we show the following:

- In Section 10.2, we present a useful technical tool we call pseudo-padded decomposition. This novel-ish type decomposition is based on $(1+\epsilon)$-approximate shortest paths. On a high-level, the algorithm creates a decomposition of strong diameter $\mathcal{D}$ where each edge is cut with probability $O\left(\frac{\alpha\ell_z}{\mathcal{D}} + \alpha\epsilon\right)$ for some parameter α (that depends on the graph). Thus, it *almost* resembles a low-diameter decomposition except that *short* edges have a much higher probability of being cut due to the additive $O(\epsilon\alpha)$. Nevertheless, it is *good enough* for many applications and will form the basis for our more sophisticated decompositions.
- In Section 10.3, we combine our pseudo-padded decomposition with the techniques by Becker, Emek, and Lenzen to obtain an LDC. Note that our algorithm works almost identically to theirs. First, we create a pseudo-padded decomposition that cuts all *long* edges with the correct probability. Then, we use their so-called *blurry ball growing* technique to refine the clusters so that the short edges are cut with the correct probability. Our construction, however, exploits specific properties of our pseudo-padded decomposition to ensure that the emerging clusters are connected and we obtain a clustering with a strong diameter.
- In Section 10.3, we then show how to extend the LDC from Section 10.3 to an LDD. In doing so, we present a general technique to turn LDC's into LDD's if the algorithm fulfills certain properties.
- In Section 10.5, we employ our weak separator from Chapter to create a better LDD for k-path separable graph. We employ a divide-and-conquer style algorithm that works in two phases. First, construct cluster *around* these separators to obtain *superclusters*. Then, we refine the *superclusters* to a proper LDD.
- In Section 10.6, we present further and more detailed related work.
- Finally, in Section 10.7, we provide a conclusion and ideas for future work.

10.2 Pseudo-Padded Decompositions Using Approximate Shortest Paths

We begin with a crucial technical theorem that will build the foundation of most our results. A key component in this construction is the use of truncated exponential variables. In particular, we will consider exponentially distributed random variables truncated to the $[0, 1]$-interval. Loosely speaking, a variable is truncated by resampling it until the outcome is in the desired interval. In the following, we will always talk about variables that are *truncated to* $[0,1]$*-interval* when we talk about truncated variables. The density function for a truncated exponential distribution with parameter $\lambda > 1$ is defined as follows:

Definition 10.2 (Truncated Exponential Distribution). *We say a random variable X is truncated exponentially distributed with parameter λ if and only if its density function is:*

$$f(x) := \frac{\lambda \cdot e^{-x\lambda}}{1 - e^{-\lambda}} \tag{10.1}$$

We write $X \sim \mathsf{Texp}(\lambda)$. Further, if $X \sim \mathsf{Texp}(\lambda)$ and $Y := \mathcal{D} \cdot X$, we write $Y \sim \mathcal{D} \cdot \mathsf{Texp}(\lambda)$.

The truncated exponential distribution is a useful tool for decompositions that has been extensively used in the past [AGG$^+$19a, Fil19, MPX13]. Using a truncated exponential distribution and $(1+\epsilon)$-approximate SetSSP computations, we prove a helpful auxiliary result, namely:

Theorem 9. *Pseudo-Padded Decomposition for General Graphs*

Let $\mathcal{D} > 0$ be a distance parameter, ϵ be an error parameter, $G := (V, E, \ell)$ a (possibly weighted) undirected graph, and $\mathcal{X} \subseteq V$ be a set of marked nodes. Suppose that for each node $v \in V$, the following two properties hold:

- **Covering Property:** There is at least one $x \in \mathcal{X}$ with $d_G(v, x) \leq \mathcal{D}$.
- **Packing Property:** There are at most τ centers $x' \in \mathcal{X}$ with $d_G(v, x') \leq 6\mathcal{D}$.

Then, for $\epsilon \in o(1/\log \tau)$ there is an algorithm that computes a series of connected clusters $\mathcal{K} = K_1, \dots, K_N$ with strong diameter $4(1+\epsilon)\mathcal{D}$ where for all nodes $v \in V$ and all $\epsilon \leq \gamma \leq \frac{1}{32}$, it holds:

$$\mathbf{Pr}[B(v, \gamma\mathcal{D}) \subset K(v)] \geq e^{-\Theta((\gamma+\epsilon)\log\tau)} - O(1/n^c).$$

Here, $K(v)$ denotes the cluster that contains v. The algorithm can be implemented with one $(1+\epsilon)$ approximate SetSSP computation and $\tilde{O}(1)$ minor aggregations.

Technically, this algorithm is a generalization of the algorithm in [Fil19] that is based on **exact** shortest path computations. This algorithm is itself derived from [MPVX15b]. Our algorithm replaces all these exact computations through $(1+\epsilon)$-approximate calculations. The main analytical challenge is carrying the resulting error ϵ through the analysis to obtain the bounds in the theorem. The same approach was already used by Becker, Lenzen, and Emek [BEL20]. However, our is more fine-grained w.r.t. to the impact of the approximation parameter ϵ.

Generally speaking, the algorithm creates a clustering that cuts each edge $z \in E$ of length greater than $\ell_z \geq \epsilon\mathcal{D}$ with probability $O\left(\frac{\ell_z \log \tau}{\mathcal{D}}\right)$. However, the bounds for smaller edges are much worse; for these edges, the error introduced by the approximate shortest path computations dominates the probabilities. Further, it is important to note that we only get non-trivial bounds for a sufficiently small ϵ because it also appears as an additive factor in the probability. In many cases, however, $\epsilon \in \Theta\left(\frac{1}{\log^2 n}\right)$ will be sufficient. Nevertheless, some applications require a very small ϵ. Recall that by Theorem 5 the algorithm can implemented in time $\tilde{O}(\epsilon^{-2} \cdot \mathrm{HD})$ time in CONGEST and $\tilde{O}(\epsilon^{-2})$ time in HYBRID, w.h.p., so it is quite sensitive to the choice of ϵ. For $\epsilon \in \Theta\left(\frac{1}{\log^2 n}\right)$, it disappears in the $\tilde{O}(\cdot)$-notation, however, smaller ϵ significantly increases the runtime. Despite all this, the theorem will turn out to be a tremendously useful building block.

Before we start with the detailed description and analysis, let's first give the high-level idea behind the construction. An intuitive way to think about the clustering process from [MPVX15b, Fil19] is as follows: Each center x draws value $\delta_x \sim \mathcal{D} \cdot \mathsf{Texp}(2 + 2\log\tau)$ wakes up at time $\mathcal{D} - \delta_x$. Then, it begins to broadcast its identifier. The spread of all centers is done in the same unit tempo. A node v joins the cluster of the *first* identifier that reaches it, breaking ties consistently. If we had $O(\mathcal{D})$ time, we could indeed implement it exactly like this. In fact, we literally do this in Chapter 3 when we compute a sparsifier of a G. However, this approach is infeasible for a general $\mathcal{D} \in \tilde{\Omega}(1)$ in weighted graphs as $\mathcal{D}$ may be arbitrarily large. Instead, we will model this intuition using a virtual super source s and shortest path computations. The source s has a weighted virtual edge (s, x) to each center $x \in \mathcal{X}$ with weight $w_x := (\mathcal{D} - \delta_x)$. Any node joins the cluster of the *last* center on its shortest path to s. With exact shortest paths, this construction preserves our intuition. Any node whose shortest path to s contains center x as its last center on the path to s would have been reached by x's broadcast first.

The statement is not that simple with approximate distances as the approximation may introduce a *detour*, so the center that minimizes $(\mathcal{D} - \delta_x) + d(x, v)$ may not actually cluster a node $v \in V$. In other words, two endpoints of a short edge (v, w) may end up in different clusters although the same center minimizes the distance to both. The nodes will only be added to same cluster, if its center's head start (which is determined by the random distance to the source) is large enough to compensate for the *detour*. This depends on the approximation and *not* an the length of the edge we consider. Nevertheless, if the error ϵ is small enough, we get similar clustering guarantees that are good enough for our purposes.

The remainder of this section is structured as follows: First, we will give a more formal description of the algorithm in Section 10.2.1 and then, in Section 10.2.2, we will prove Theorem 9.

10.2.1 Algorithm Description

We now describe the algorithm promised by Theorem 9 in more detail. Note that we will express the algorithm in terms of approximate SetSSP computations and minor aggregations. With the help of Theorem 5, we will later derive the runtimes for CONGEST and HYBRID. The algorithm works in the following 4 synchronized steps:

(Step 1) Draw random distances: In the first step, for each center $x \in \mathcal{X}$, we independently draw a value $\delta_x \in [0, \mathcal{D}]$ from the truncated exponential distribution with parameter $2 + 2\log(\tau)$. We assume that τ (or some upper bound thereof) is known to each node. Formally, we have

$$\delta_x \sim \mathcal{D} \cdot \mathsf{Texp}(2 + 2\log\tau)$$

We call δ_x the offset parameter of center x. Note that $\mathsf{Texp}(2 + 2\log\tau)$ returns a continuous random variable. So, we implicitly round the value to the next value that can be encoded in $c\log n$ bits for some $c \geq 1$ in order to send it around in a single message. Therefore, δ_x differs from $\mathcal{D} \cdot \mathsf{Texp}(2 + 2\log\tau)$ by at most $\mathcal{D}/n^c$.

(Step 2) Add a (virtual) super-source s: In next step, we add a virtual node s to G and obtain the graph $G_s := (V \cup \{s\}, E \cup E_s, w_s)$. The source s has a weighted virtual edge $\{s, x\} \in E_s$ to each center $x \in \mathcal{X}$ with weight $w_x := (\mathcal{D} - \delta_x)$. All other weights and edges remain unchanged.

(Step 3) Construct an approximate shortest-path tree T_s: Next, we perform an $(1+\epsilon)$-approximate SSSP algorithm in G_s from node s.

Recall that our $\epsilon > 0$ is the desired error parameter for the pseudo-padded decomposition. Let T be the (approximate) shortest-path tree computed by the SSSP algorithm, s.t., it holds:

$$d_T(v, s) \leq (1+\epsilon) d_{G_s}(v, s)$$

(Step 4) Join Cluster of Closest Center: Finally, a vertex v joins the cluster of the center $x \in \mathcal{X}$, s.t.,

$$x := \arg\min_{x' \in \mathcal{X}} \{\mathcal{D} - \delta_{x'} + d_T(x', v)\}$$

It is easy to verify that this is the first center on the path from s to v. Thus, it only remains to inform each node about their clusters identifier. This can be done via the $\mathsf{AncestorSum}$ primitive, which acts as a broadcast if only the root has an input value. Using this primitive, each center $x \in \mathcal{X}$ broadcasts its identifier and its distance to s to each node in its subtree $T(x)$.

Finally, note that this construction ensures that for all pairs $v, w \in C_x$, there is a path in C_x that connects them.

10.2.2 Analysis

In this section, we will prove Theorem 9. To this end, we must show that the algorithm a) can be implemented with *one* approximate set-source SetSSP computation, b) creates clusters of strong diameter $(1+\epsilon)4\mathcal{D}$, and c) fulfills the padding guarantee from Theorem 9. We begin by proving the proclaimed complexity, arguably easiest to show. This can be derived directly from the description of the algorithm. The algorithm requires only a single approximate shortest-path computation as Steps 1, 2, and 4 are purely local operations. Therefore, it only remains to show the other two properties. We begin with the strong diameter and show that for all vertices within a cluster, there is a path of length at most $(1+\epsilon)4\mathcal{D}$ *within* the cluster. Formally, we want to show:

Lemma 10.1 (Cluster Diameter)**.** *Every non-empty cluster C_x created by the algorithm has strong diameter at most $(1+\epsilon)4\mathcal{D}$, i.e., for two vertices $v, w \in C_x$, it holds:*

$$d_{C_x}(v,w) \leq 4(1+\epsilon)\mathcal{D}$$

For the proof, we first recall how the clusters are built. Each vertex joins a cluster C_x of some center $x \in \mathcal{X}$ if and only if there is a path in the approximate shortest path tree T_s where x is the last hop before the root s. Thus, each vertex has a path to x that is fully contained in the cluster. That means, for two vertices $v, w \in C_x$, there is a path from v to w via x that is entirely contained in the cluster as all edges are undirected. This implies that:

$$d_{C_x}(v,w) \leq 2 \cdot \max_{u \in C_x} d_{T_s}(u,x) \tag{10.2}$$

Therefore, it remains to bound the maximal distance between a node and its cluster center. Here, we see that it holds for all $x \in \mathcal{X}$:

Claim 13. $\max_{u \in C_x} d_{T_s}(u,x) \leq (1+\epsilon)2\mathcal{D}$

Proof. Consider node $u \in V$ that maximizes the distance to x in T_s. As x is the penultimate node on the path to s in T_s, the distance to x is upper bounded by the distance to s. It holds:

$$d_{T_s}(s,u) := w(s,x) + d_{T_s}(x,u) := (\mathcal{D} - \delta_x) + d_{T_s}(x,u) \geq d_{T_s}(x,u) \tag{10.3}$$

The inequality follows because $\delta_x \in [0,\mathcal{D}]$ per construction and therefore $(\mathcal{D} - \delta_x)$ is non-negative. Further, by the covering property required in Theorem 9, we know that there is at least one center $x' \in \mathcal{X}$ in the distance at most $\mathcal{D}$ to v. Note that the distance between x' and s is also at most $\mathcal{D}$ as $\delta_{x'} \geq 0$ by definition. Thus, by the triangle inequality, it holds:

$$d_G(s,u) \leq d_G(s,x') + d_G(x',u) := (\mathcal{D} - \delta_{x'}) + d_G(x',u) \leq \mathcal{D} + \mathcal{D} = 2\mathcal{D}.$$

Combining these two observations with the approximation guarantee from the underlying SSSP gives us the following upper bound on the distance to the cluster center:

$$\begin{aligned} d_{T_s}(x,u) &\leq d_{T_s}(s,u) && \triangleright \textit{By Ineq.}\ (10.3)\\ &\leq (1+\epsilon)d_G(s,u) && \triangleright \textit{As}\ d_T(s,u) \leq (1+\epsilon)\cdot d(s,u)\\ &\leq (1+\epsilon)2\mathcal{D} && \triangleright \textit{By Ineq.}\ (10.4) \end{aligned}$$

This concludes the proof! □

Together with Inequality (10.2), this claim directly implies Lemma 10.1 and our algorithm is guaranteed to create clusters of strong diameter $4(1+\epsilon)\mathcal{D}$. We continue with the padding property promised by Theorem 9. To this end, we need to show that all nodes that are *close* to a given vertex $v \in V$ are in the same cluster with some non-trivial probability. In fact, we will show a slightly stronger statement that we will reuse in later algorithms. We show:

Lemma 10.2 (Padding Property). *Consider node $v \in V$ clustered by center $x_v \in \mathcal{X}$ and a parameter $\epsilon \leq \gamma \leq \frac{1}{8}$. Further, let $P_{x_v} = (v_1, \dots, v_\ell)$ be the* ***exact*** *shortest path from x_v to v. Then, for each $v_i \in P_{x_v}$, the ball $B_G(v_i, \gamma\mathcal{D})$ is fully contained in C_{x_v} with probability at least*

$$\mathbf{Pr}\left[\bigcup_{v_i \in P_{x_v}} B_G(v_i, \gamma\mathcal{D}) \subset C_{x_v}\right] \geq e^{-16(\gamma+5\epsilon)\log\tau} - \frac{160\log\tau}{n^c}$$

First, for simpler notation, we introduce the term $d(s, x, v)$, which is a shorthand for the distance between v and s via the center $x \in \mathcal{X}$ in graph G_s. It holds:

$$d(s, x, v) := (\mathcal{D} - \delta_x) + d(x, v).$$

Note that this definition is *only* based on the properties of the input graph G and the random distances $\mathcal{D}$. Further, we let N_v be the set of centers $x \in \mathcal{X}$ in the distance at most $6\mathcal{D}$ to node $v \in V$. Note that these include the centers which can potentially cluster v and for which there is a non-zero probability that C_x intersects $B(v, \gamma\mathcal{D})$. In other words, these are all the centers of all clusters that (potentially) neighbor the cluster that contains v. This can be verified by the following argument: Let $v_i \in P_{x_v}$ an node on the shortest path from v to x_v and let $x \in \mathcal{X}$ be a center that can potentially cluster a node $w \in B(v_i, \gamma\mathcal{D})$. Any node v_i on the path is in distance at most $(1+\epsilon)2\mathcal{D}$ to v; any node $w \in B(v_i, \gamma\mathcal{D})$ is in distance at most $(1/8) \cdot \mathcal{D}$ to w as $\gamma \leq 1/8$; and finally every center x that can cover w is in distance $(1+\epsilon)2\mathcal{D}$ to x. Summing all these distances up and using that $\epsilon \leq \gamma \leq 1/8$ gives us:

$$\begin{aligned} d_G(v, x) &\leq d_G(v, w) + d_G(w, b) + d_G(w, x) \\ &\leq (1+\epsilon)2\mathcal{D} + \gamma\mathcal{D} + (1+\epsilon)2\mathcal{D} \\ \text{As } \epsilon \leq \gamma \leq 1/8: & \\ &\leq (1+1/8)2\mathcal{D} + (1/8)\mathcal{D} + (1+1/8)2\mathcal{D} \\ &\leq 6\mathcal{D} \end{aligned}$$

Recall that by the *packing property* in Theorem 9, we have $|N_v| \leq \tau$. We continue with the definition of the central concept of our proof, the random shift Υ_i for each center $x_i \in N_v$. It is defined as follows:

Definition 10.3 (Random Shift). *Let $v \in V$ be a node and let $N_v := \{x \in \mathcal{X} \mid d(x, v) \leq 6\mathcal{D}\}$. For each $x \in N_v$, define the random shift $\Upsilon_v^{(x)}$ as follows:*

$$\Upsilon_v^{(x)} := (\mathcal{D} - \delta_x) + d_G(x, v) - \min_{x' \in N_v \setminus \{x\}} ((\mathcal{D} - \delta_{x'}) + d_G(x', v))$$

These values will help us to quantify how close a center is to node $v \in V$ compared to the other centers. From the definition of $\Upsilon_v^{(x)}$, it follows that for all $x' \in \mathcal{X} \setminus \{x\}$, it holds:

Claim 14. *For $x' \in N_v$, it holds $d_G(v,x,s) \leq d_G(v,x',s) + \Upsilon_v^{(x)}$.*

Proof. Using the definitions of $d_G(v,x,s)$ and $\Upsilon_v^{(x)}$, we get:

$$\begin{aligned} d_G(v,x,s) &= (\mathcal{D} - \delta_x) + d(x,v) \\ &= (\mathcal{D} - \delta_{x'}) + d(x',v) + ((\mathcal{D} - \delta_x) + d(x,v) - ((\mathcal{D} - \delta_{x'}) + d(x',v))) \\ &\leq d_G(v,x',s) + \left((\mathcal{D} - \delta_x) + d(x,v) - \min_{x'' \in \mathcal{X} \setminus \{x\}} \{(\mathcal{D} - \delta_{x''}) + d(x'',v)\} \right) \\ &:= d_G(v,x',s) + \Upsilon_v^{(x)} \end{aligned}$$

This proves the claim. □

We furthermore need the following lemma.

Lemma 10.3. *Let $v \in V$ be a node and let P_x be the **exact** shortest path to center $x \in \mathcal{X}$ in G. Further, let $\mathcal{X}' \subseteq \mathcal{X}$ be the set of centers that can potentially cluster a node $u \in P$. Define the values $\Upsilon_v^{(x)}$ and $\Upsilon_u^{(x)}$ as follows:*

$$\Upsilon_v^{(x)} := (\mathcal{D} - \delta_x) + d_G(x,v) - \min_{x' \in \mathcal{X}' \setminus \{x\}} ((\mathcal{D} - \delta') + d_G(x',v)) \tag{10.4}$$

$$\Upsilon_u^{(x)} := (\mathcal{D} - \delta_x) + d_G(x,u) - \min_{x' \in \mathcal{X}' \setminus \{x\}} ((\mathcal{D} - \delta') + d_G(x',u)) \tag{10.5}$$

Then, it holds $\Upsilon_u^{(x)} \leq \Upsilon_v^{(x)}$.

Proof. First, for convenience, we define:

$$x_u := \arg\min_{x' \in \mathcal{X}' \setminus \{x\}} ((\mathcal{D} - \delta') + d_G(x',v)) \,. \tag{10.6}$$

Next, we add $(d_G(u,v) - d_G(u,v)) = 0$ to $\Upsilon_u^{(x)}$ and obtain:

$$\begin{aligned} \Upsilon_u^{(x)} &= (\mathcal{D} - \delta_i) + d_G(x_i,u) - ((\mathcal{D} - \delta_u) + d_G(x_u,u)) \\ &= (\mathcal{D} - \delta_i) + d_G(x_i,u) - ((\mathcal{D} - \delta_u) + d_G(x_u,u)) + (d_G(u,v) - d_G(u,v)) \\ &= (\mathcal{D} - \delta_i) + d_G(x_i,u) + d_G(u,v) - ((\mathcal{D} - \delta_u) + d_G(x_u,u) + d_G(u,v)) \end{aligned}$$

Now, we use the fact that u is an ancestor of v in the *exact* shortest path P. By this fact, it holds:

$$d_G(x_i,v) = d_G(x_i,v) + d_G(u,v) \tag{10.7}$$

Otherwise, there would be a shorter path from v to x. Therefore,

$$\Upsilon_u^{(x)} \underset{(10.7)}{=} (\mathcal{D} - \delta_i) + d_G(x_i,v) - ((\mathcal{D} - \delta_u) + d_G(x_u,u) + d_G(u,v))$$

By the triangle inequality, we therefore obtain the following inequality:

$$d_G(x_u, u) + d_G(u, v) \geq d_G(x_u, v) \tag{10.8}$$

Putting this insight back into the formula gives us:

$$\Upsilon_u^{(x)} \underset{(10.8)}{\leq} (\mathcal{D} - \delta_i) + d_G(x_i, v) - ((\mathcal{D} - \delta_u) + d_G(x_u, v))$$

By definition, it holds:

$$((\mathcal{D} - \delta_u) + d_G(x_u, v)) \geq \min_{x' \in \mathcal{X}' \setminus \{x\}} ((\mathcal{D} - \delta_u) + d_G(x', v)) \tag{10.9}$$

Thus, we have:

$$\Upsilon_u^{(x)} \underset{(10.9)}{\leq} (\mathcal{D} - \delta_i) + d_G(x_i, v) - \min_{x' \in \mathcal{X}' \setminus \{x\}} ((\mathcal{D} - \delta_u) + d_G(x', v)) = \Upsilon_v^{(x)}$$

This proves the lemma. □

Note that, when using exact distances, the center that minimizes $\Upsilon_v^{(x)}$ will add v to its cluster. However, this is not necessarily true when using approximate distances, as the approximation error can make another center appear closer. Nevertheless, we can show that for a big enough difference $\Upsilon_v^{(x)}$, the center x clusters node v and, more importantly, also clusters all nodes close to v. More precisely, it holds:

Lemma 10.4. *Let $x \in N_v$ be a center in $\mathcal{X}$ in the distance at most $6\mathcal{D}$ to v. Further, let $P := (v_1, \ldots, v_\ell)$ with $v_1 = x$ and $v_\ell = v$ be the **exact** shortest path from x to v in G.*

Consider a node $v_i \in P$ and a node $u \in B(v_i, \gamma\mathcal{D})$. Suppose it holds

$$\Upsilon_v^{(x)} < -(2\gamma + 10\epsilon)\mathcal{D}. \tag{10.10}$$

Then this implies $v_i, u \in C_x$. In particular, this means that for all $v_i \in P$, it holds:

$$\mathbf{Pr}[B(v_i, \gamma\mathcal{D}) \subset C_x] \geq \mathbf{Pr}\Big[\Upsilon_v^{(x)} \leq -(2\gamma + 10\epsilon) \cdot \mathcal{D}\Big]$$

Proof. Let x_u be any center that can potentially cluster node u. By Claim 13, each vertex joins the cluster of a center at a distance at most $(1+\epsilon)2\mathcal{D}$. Therefore, the distances between v and v_i and u and x_u is at most $(1+\epsilon)2\mathcal{D}$. As $u \in B(v, \gamma\mathcal{D})$, by the triangle inequality, the distance between x_u and v is at most

$$\begin{aligned} d_G(v, x_u) &\leq d_G(v, v_i) + d_G(v_i, u) + d(u, x_u) \\ &\leq (1+\epsilon)2\mathcal{D} + \gamma\mathcal{D} + (1+\epsilon)2\mathcal{D} \\ &\leq (1+\epsilon)4\mathcal{D} + \gamma\mathcal{D} \leq 6\mathcal{D} \end{aligned}$$

Therefore, $x_u \in N_v$.

We want to show that for $\Upsilon_v^{(x)} \leq -(2\gamma + 10\epsilon)\mathcal{D}$, center x is closer to u than x_u **even** when using approximate distances. The idea behind the proof is, loosely speaking, that for a big enough value of $\Upsilon_v^{(x)}$ the error introduced by the approximate SSSP is *canceled out* in some sense. Suppose for contradiction that x_u (and *not* x) adds node u to its cluster, i.e., it holds $u \in C_{x_u}$. This only happens if and only if x_u is the penultimate node on the approximate shortest path to s. Or, equivalently, is the penultimate node on the exact path in the tree T_s Formally, it holds:

$$d_{T_s}(s,u) = d(s,x_u) + d_{T_s}(x_u,u) \geq d(s,x_u) + d_{G_s}(x_u,u) = d(s,x_u,u)$$

On the other hand, using approximation guarantee and the triangle inequality, we see that the following holds:

$$\begin{aligned} d_{T_s}(s,u) &\leq (1+\epsilon) \cdot d_{G_s}(s,u) \\ &\leq d_{G_s}(s,x,u) + \epsilon \cdot d_{G_s}(s,x,u) \\ &\leq d_{G_s}(s,x,v_i) + d_{G_s}(v_i,u) + \epsilon \cdot d_{G_s}(s,x,u) \end{aligned}$$

By Claim 14, we furthermore get:

$$\begin{aligned} d_{T_s}(s,u) &\leq \left(d_{G_s}(s,x_u,v_i) + \Upsilon_{v_i}^{(x)}\right) + d_{G_s}(u,v_i) + \epsilon \cdot d_{G_s}(s,x,u) \\ &\leq \left(d_{G_s}(s,x_u,u) + d_{G_s}(u,v_i) + \Upsilon_{v_i}^{(x)}\right) + d_{G_s}(u,v_i) + \epsilon \cdot d_{G_s}(s,x,u) \\ &\leq d_{G_s}(s,x_u,u) + 2d_{G_s}(u,v_i) + \epsilon \cdot d_{G_s}(s,x,u) + \Upsilon_{v_i}^{(x)} \end{aligned}$$

As v_i lies on the exact shortest path from v to x, it holds by Lemma 10.3 that $\Upsilon_{v_i}^{(x)} \leq \Upsilon_v^{(x)}$ and therefore:

$$d_{T_s}(s,u) \leq d_{G_s}(s,x_u,u) + 2d_{G_s}(u,v_i) + \epsilon \cdot d_{G_s}(s,x,u) + \Upsilon_v^{(x)}$$

Now, we bound $d_{G_s}(s,x,u)$. To this end, recall that $d(s,x) = \mathcal{D} - \delta_x \leq \mathcal{D}$. Further, since $x \in N_v$, we have $d(x,v_i) \leq d(x,v) \leq 6\mathcal{D}$. Finally, by definition, we have $d(v_i,u) = \gamma\mathcal{D}$. Thus, combining all these facts, it holds:

$$\begin{aligned} d_{G_s}(s,x,u) &:= d_{G_s}(s,x) + d_{G_s}(x,u) \\ &\leq d_{G_s}(s,x) + d_{G_s}(x,v_i) + d_{G_s}(v_i,u) \\ &\leq \mathcal{D} + 6\mathcal{D} + \gamma\mathcal{D} \leq 8\mathcal{D} \end{aligned}$$

Now use our assumptions and see:

$$
\begin{aligned}
d_{T_s}(s,u) &\leq d_{G_s}(s,x_u,u) + 2d_{G_s}(u,v_i) + \epsilon \cdot d_{G_s}(s,x,u) + \Upsilon_v^{(x)} \\
\text{As } d_{G_s}(s,x,u) \leq 8\mathcal{D}: & \\
&\leq d_{G_s}(s,x_u,u) + 2d_{G_s}(u,v) + \epsilon \cdot 8\mathcal{D} + \Upsilon_v^{(x)} \\
\text{As } d_{G_s}(v,u) \leq \gamma\mathcal{D}: & \\
&\leq d_{G_s}(s,x_u,u) + 2\gamma\mathcal{D} + \epsilon \cdot 8\mathcal{D} + \Upsilon_v^{(x)} \\
\text{As } \Upsilon_v^{(x)} < -(2\gamma + 10\epsilon)\mathcal{D}: & \\
&\leq d_{G_s}(s,x_u,u) + 2\gamma\mathcal{D} + \epsilon \cdot 8\mathcal{D} - 2\gamma\mathcal{D} - \epsilon \cdot 10\mathcal{D} \\
&< d_{G_s}(s,x_u,u)
\end{aligned}
$$

Thus, it follows that:

$$
d_{T_s}(s,u) \underset{!}{<} d_{G_s}(s,x_u,u) \leq d_{T_s}(s,u)
$$

This is a contradiction. As this holds for any possible choice $x_u \in N_v$, none of them add u to their cluster. Therefore u must be part of C_x as claimed and the lemma follows. □

This lemma tells us that — if the random shift δ_x is big enough — a center $x \in N_v$ will cluster all nodes that are *close* to its shortest path to v. Again, note that this statement is stronger than what we need for a pseudo-padded decomposition, but we show it in this generality to reuse it later in the next section. In the remainder of this section, we will prove that with non-trivial probability, there is a center $x_i \in N_v$ with a large enough shift. Concretely, we show the following lemma:

Lemma 10.5 (Random Shift with Approximate Distances). *Suppose that*

$$
\begin{aligned}
&\gamma \leq 1/8 \\
&\epsilon \leq \min\left\{\frac{1}{40}, \frac{1}{20 \cdot (2\log\tau + 2)}\right\}
\end{aligned}
$$

Then, it holds

$$
\mathbf{Pr}[\exists x_i \in N_v : \Upsilon_i \leq -(2\gamma + 10\epsilon) \cdot \mathcal{D}] \geq e^{-16(\gamma+\epsilon)\log\tau} - \frac{160\log(\tau)}{n^c}
$$

The full proof is presented below. It is based on the proof of Theorem 1 in [Fil19] that proves a similar statement but assumes exact shortest path computations. The main difficulty is quantifying the effect of approximation parameter ϵ.

We conclude this section by proving Lemma 10.2 using Lemma 10.4 and Lemma 10.5. Note that, by Lemma 10.4, it holds for all $v_j \in P_{x_v}$ that

$$\mathbf{Pr}[B(v_j, \gamma\mathcal{D}) \subset C_{x_v}] \geq \mathbf{Pr}[\exists x_i \in N_v : \Upsilon_i \leq -(2\gamma + 10\epsilon) \cdot \mathcal{D}]$$

Further, by Lemma 10.5, we have

$$\mathbf{Pr}[\exists x_i \in N_v : \Upsilon_i \leq -(2\gamma + 10\epsilon) \cdot \mathcal{D}] \geq e^{-16(\gamma+5\epsilon)\log\tau} + \frac{160\log\tau}{n^c}$$

This proves Lemma 10.2. Together with the initial observation that the algorithm can be implemented with a *single* approximate shortest path computation on a set-source, this implies Theorem 9.

Proof of Lemma 10.5

In this section, we will prove Lemma 10.5 using techniques and arguments from [Fil19] where a similar statement was shown for *exact* distance computations. In fact, we follow the analysis almost verbatim and only adapt the arguments that do not generally hold for approximate distances. The main technical difficulties are in the facts that a) we need to carry the approximation error ϵ through the calculations and b) that the triangle inequality does not hold for approximate distances and, therefore, a node may not be added to the cluster of the center on the exact shortest path to virtual source s.

Condition on the event that there are τ centers in N_v and let $N_v = \{x_1, x_2, \dots, x_\tau\}$. Until now, we have always considered the rounded random variables $\delta_{x_1}, \dots, \delta_{x_\tau}$ that can be encoded in $c\log n$ bits. These are the variables that will be used by the approximate shortest path algorithm to compute the clusters. In this section, we will work with actual continuous values, i.e., the actual values returned by sampling the truncated distribution. For each center $x_i \in N_v$, we defined the random shift as

$$\Upsilon'_i := (\mathcal{D} - \delta'_i \cdot \mathcal{D}) + d_G(x_i, v) - \min_{x_j \in N_v \setminus \{x_i\}} \big((\mathcal{D} - \delta'_j \cdot \mathcal{D}) + d_G(x_j, v)\big)$$

As we round them up to be encodable in $c\log n$ bits for some $c \geq 1$, it holds for all $x_i \in N_v$ that

$$\delta_{x_i} - \delta'_{x_i} \cdot \mathcal{D} \leq \frac{1}{n^c} \tag{10.11}$$

$$(\mathcal{D} - \delta'_i \cdot \mathcal{D}) + d_G(x_i, v) - (\mathcal{D} - \delta_i) + d_G(x_i, v) \leq \frac{1}{n^c} \tag{10.12}$$

$$|\Upsilon'_i - \Upsilon_i| \leq \frac{1}{n^c} \tag{10.13}$$

Thus, the error introduced by the rounding is negligible. Nevertheless, we need to carry it through.

Denote by $\mathcal{F}_i$ the event that x_i is the truly closest center to v. In other words, the value Υ'_i is non-positive and if there is another center with the same distance, it has a lower identifier, i.e., we have

$$\mathcal{F}_i := \left\{ x_i = \arg\min_{x_i \in N_v} \Upsilon_i \leq 0 \right\} \tag{10.14}$$

Note that the tie-breaker only comes into effect when there two centers $x_, x_j$ with $\Upsilon_i = \Upsilon_j = 0$. With exact distance computations, the minimal center will cluster v as it is on the shortest path to supersource s. This is not necessarily true when using approximate distances and rounding. As we always round down, it holds:

$$\mathbf{Pr}\left[\Upsilon'_i < -\frac{1}{n^c}\right] \leq \mathbf{Pr}[\mathcal{F}_i] \leq \mathbf{Pr}\left[\Upsilon'_i \leq \frac{1}{n^c}\right] \tag{10.15}$$

Further, let $\epsilon_l = \epsilon + 1/n^c$ and $\epsilon_u := 1/n^c$ denote by $\mathcal{C}_i$ the event that

$$\mathcal{C}_i := \{\Upsilon'_i \in (-2 \cdot (\gamma + 5\epsilon_l) \cdot \mathcal{D}, \epsilon_u]\}.$$

In other words, a center x_i is the closest center but not by a lot. This is a *bad* event as it implies that the ball $B(v, \gamma\mathcal{D})$ may not be added to C_{x_i}. Perhaps not even v is added. On the flip side, if *none* of these *bad* events $\mathcal{C}_1, \ldots, \mathcal{C}_\tau$ happen, there *must* be a center with a big enough shift. Recall that we want to show that:

$$\mathbf{Pr}[\exists x_i \in N_v : (\Upsilon_i \leq -2 \cdot (\gamma + 5\epsilon) \cdot \mathcal{D})] \geq e^{-16(\gamma+5\epsilon)\log\tau} - \frac{160\log\tau}{n^c}$$

It holds:

$$\mathbf{Pr}[\forall x_i \in N_v : \Upsilon_i \geq -2 \cdot (\gamma + 5\epsilon) \cdot \mathcal{D}]$$

As all $\mathcal{F}_i$ disjoint:

$$= \sum_{i=1}^{\tau} \mathbf{Pr}[\mathcal{F}_i] \cdot \mathbf{Pr}[\forall x_i \in N_v : \Upsilon_i \geq -2 \cdot (\gamma + 5\epsilon) \cdot \mathcal{D} \mid \mathcal{F}_i]$$

As $\mathcal{F}_i$ implies Υ_i is minimal:

$$= \sum_{i=1}^{\tau} \mathbf{Pr}[\mathcal{F}_i] \cdot \mathbf{Pr}[\Upsilon_i \geq -2 \cdot (\gamma + 5\epsilon) \cdot \mathcal{D} \mid \mathcal{F}_i]$$

$$= \sum_{i=1}^{\tau} \mathbf{Pr}[\Upsilon_i \geq -2 \cdot (\gamma + 5\epsilon) \cdot \mathcal{D} \cap \mathcal{F}_i]$$

By Ineq. (10.14):

$$= \sum_{i=1}^{\tau} \mathbf{Pr}[\Upsilon_i \geq -2 \cdot (\gamma + 5\epsilon) \cdot \mathcal{D} \cap \{\Upsilon'_i \leq 1/n^c\}]$$

As $|\Upsilon_i - \Upsilon'_i| \leq 1/n^c$:

$$\leq \sum_{i=1}^{\tau} \mathbf{Pr}\big[\Upsilon'_i \in \big(-2 \cdot (\gamma + 5\epsilon + 1/n^c) \cdot \mathcal{D}, 1/n^c\big]\big]$$

$$\leq \sum_{i=1}^{\tau} \mathbf{Pr}\big[\Upsilon'_i \in \big(-2 \cdot (\gamma + 5\epsilon_l) \cdot \mathcal{D}, \epsilon_u\big]\big]$$

$$= \sum_{i=1}^{\tau} \mathbf{Pr}[\mathcal{C}_i]$$

Using this insight, we note that the following equality holds:

$$\mathbf{Pr}[\exists x_i \in N_v : \Upsilon_i \leq -2 \cdot (\gamma + 5\epsilon) \cdot \mathcal{D}] \geq 1 - \sum_{i=1}^{\tau} \mathbf{Pr}[\mathcal{C}_i]. \tag{10.16}$$

In the following, we will bound $\mathbf{Pr}[\mathcal{C}_i]$ for a fixed x_i and show the following.

Lemma 10.6. *Let* $\lambda := 2\log\tau + 2$. *Then, for all* $i \leq \tau$, *it holds:*

$$\mathbf{Pr}[\mathcal{C}_i] \leq \left(1 - e^{-2(\gamma+5\epsilon_l+\epsilon_u)\cdot\lambda}\right) \cdot e^{2\lambda\epsilon_u} \cdot \left(\mathbf{Pr}[\mathcal{F}_i] + \frac{1}{e^{\lambda}-1}\right)$$

Proof. For easier notation, we drop the index and denote $x = x_i$ to simplify notation and analogously fix $\mathcal{C} := \mathcal{C}_i, \mathcal{F} := \mathcal{F}_i$, and $\delta' := \delta'_{x_i}$. Consider a subset of τ centers $\mathcal{X}' := \{x_1, \ldots, x_\tau\}$ and let $\mathcal{Z} := \{\delta_1, \ldots, \delta_\tau\}$ be a realization of their random shifts. Further, we define the value

$$\rho_{\mathcal{Z}} := \frac{1}{\mathcal{D}} \cdot \left(d_G(x,v) + \max_{j<\tau}\left\{\delta'_{x_j} \cdot \mathcal{D} - d_G(x_j, v)\right\}\right)$$

Further, define $\Upsilon'_{\mathcal{Z}} = \mathcal{D} \cdot \rho_{\mathcal{Z}} - \mathcal{D} \cdot \delta'$ as the difference between x and the closest center. Note that $\Upsilon'_{\mathcal{Z}}$ is a random variable that only depends on the shift δ (as everything else is fixed by conditioning on $\mathcal{Z}$). To prove the Lemma, we will use that the following holds for any value $\alpha \geq 0$:

$$\mathbf{Pr}[\Upsilon'_{\mathcal{Z}} \leq -\alpha\mathcal{D}] = \mathbf{Pr}[\mathcal{D}(\rho_{\mathcal{Z}} - \delta') \leq -\alpha\mathcal{D}] = \mathbf{Pr}[\delta' \geq \rho_{\mathcal{Z}} + \alpha]$$

We will now use the law of total probability to prove the lemma. Denote by f the density function of the distribution over all possible values of δ'. By the law of total probability, it holds:

$$\begin{aligned}
\mathbf{Pr}[\mathcal{C} \mid \mathcal{Z}] &:= \int_{y=0}^{1} \mathbf{Pr}[\mathcal{C}_{\mathcal{Z}} \mid \delta = y] f(y) dy \\
&\leq \int_{y=\rho-\epsilon_u}^{\min\{1,\rho+2(\gamma+5\epsilon_l)\}} \mathbf{Pr}[\mathcal{C}_{\mathcal{Z}} \mid \delta = y] f(y) dy \\
&\leq \int_{y=\rho-\epsilon_u}^{\min\{1,\rho+2(\gamma+5\epsilon_l)\}} f(y) dy \\
&= \int_{\rho-\epsilon_u}^{\min\{1,\rho+2(\gamma+5\epsilon_l)\}} \frac{\lambda \cdot e^{-\lambda y}}{1 - e^{-\lambda}} dy
\end{aligned}$$

We can simplify this statement using some fundamental calculations and the definition of the truncated exponential function. It holds:

Claim 15. *Consider a random variable $\delta' \sim \mathsf{Texp}(\lambda)$ drawn from a truncated exponential distribution with parameter $\lambda > 0$. For values $\rho, \gamma', \epsilon' > 0$, it holds:*

$$\int_{\rho-\epsilon'}^{\min\{1,\rho+\gamma'\}} \frac{\lambda \cdot e^{-\lambda y}}{1-e^{-\lambda}} dy \leq \left(1 - e^{-\lambda(\gamma'+\epsilon')}\right) \cdot e^{2\epsilon'\lambda} \cdot \left(\mathbf{Pr}[\delta \geq \rho + \epsilon'] + \frac{1}{e^{\lambda}-1}\right)$$

Proof. The proof follows directly from the definition of δ. First, we note that it holds that

$$\mathbf{Pr}[\delta > \rho + \epsilon'] = \int_{\rho+\epsilon'}^{1} \frac{\lambda \cdot e^{-\lambda y}}{1-e^{-\lambda}} dy = \frac{e^{-(\rho+\gamma')\cdot\lambda} - e^{-\lambda}}{1-e^{-\lambda}} . \tag{10.17}$$

On the other hand, it holds:

$$\begin{aligned}
\mathbf{Pr}[\rho - \epsilon' \leq \delta \leq \rho + \gamma'] &= \int_{\rho-\epsilon'}^{\min\{1,\rho+\gamma'\}} \frac{\lambda \cdot e^{-\lambda y}}{1-e^{-\lambda}} dy \\
&\leq \frac{e^{-(\rho-\epsilon')\cdot\lambda} - e^{-\left(\rho+\gamma'\right)\cdot\lambda}}{1-e^{-\lambda}} \\
&= \frac{e^{-(\rho-\epsilon')\cdot\lambda} - e^{-\left(\rho-\epsilon'+\epsilon'+\gamma'\right)\cdot\lambda}}{1-e^{-\lambda}} \\
&= \frac{e^{-(\rho-\epsilon')\cdot\lambda} - e^{-\left(\rho-\epsilon'\right)\cdot\lambda-(\epsilon'+\gamma')\cdot\lambda}}{1-e^{-\lambda}} \\
&= \left(1 - e^{-(\gamma'+\epsilon')\cdot\lambda}\right) \cdot \frac{e^{-(\rho-\epsilon')\cdot\lambda}}{1-e^{-\lambda}} \\
&= \left(1 - e^{-(\gamma'+\epsilon')\cdot\lambda}\right) \cdot \frac{e^{-(\rho+\epsilon'-2\epsilon')\cdot\lambda}}{1-e^{-\lambda}} \\
&= \left(1 - e^{-(\gamma'+\epsilon')\cdot\lambda}\right) \cdot e^{2\epsilon'\lambda} \cdot \frac{e^{-(\rho+\epsilon')\cdot\lambda}}{1-e^{-\lambda}} \\
&= \left(1 - e^{-(\gamma'+\epsilon')\cdot\lambda}\right) \cdot e^{2\epsilon'\lambda} \cdot \frac{e^{-(\rho+\epsilon')\cdot\lambda} - e^{-\lambda} + e^{-\lambda}}{1-e^{-\lambda}} \\
&= \left(1 - e^{-(\gamma'+\epsilon')\cdot\lambda}\right) \cdot e^{2\epsilon'\lambda} \cdot \left(\frac{e^{-(\rho+\epsilon')\cdot\lambda} - e^{-\lambda}}{1-e^{-\lambda}} + \frac{e^{-\lambda}}{1-e^{-\lambda}}\right) \\
&= \left(1 - e^{-(\gamma'+\epsilon')\cdot\lambda}\right) \cdot e^{2\epsilon'\lambda} \cdot \left(\mathbf{Pr}[\delta > \rho + \epsilon'] + \frac{e^{\lambda}e^{-\lambda}}{e^{\lambda}(1-e^{-\lambda})}\right) \\
&= \left(1 - e^{-(\gamma'+\epsilon')\cdot\lambda}\right) \cdot e^{2\epsilon'\lambda} \cdot \left(\mathbf{Pr}[\delta > \rho + \epsilon'] + \frac{1}{e^{\lambda}-1}\right)
\end{aligned}$$

Thus, the claim follows. □

Now, we set $\gamma' = 2\gamma + 10\epsilon_l$ and $\epsilon' = \epsilon_u$ in our formula. We get:

$$\mathbf{Pr}[\mathcal{C} \mid \mathcal{Z}] \leq \left(1 - e^{-2\lambda(\gamma+5\epsilon_l+\epsilon_u)}\right) \cdot e^{2\lambda\epsilon_u} \cdot \left(\mathbf{Pr}[\delta \geq \rho + \epsilon_u] + \frac{1}{e^{\lambda}-1}\right) \tag{10.18}$$

To conclude the proof, denote by f the density function of the distribution over all possible values of $\mathcal{Z}$. Using the law of total probability, we can bound the probability for event $\mathcal{C}$ as follows:

$$\begin{aligned}
\mathbf{Pr}[\mathcal{C}] &= \int_{\mathcal{Z}} \mathbf{Pr}[\mathcal{C} \mid \mathcal{Z}] \cdot f(\mathcal{Z})\, d\mathcal{Z} && (10.19)\\
&\leq \int_{\mathcal{Z}} \left(1 - e^{-2\lambda(\gamma+5\epsilon_l+\epsilon_u)}\right) \cdot e^{2\lambda\epsilon_u} \cdot \left(\mathbf{Pr}[\delta \geq \rho + \epsilon_u] + \frac{1}{e^{\lambda}-1}\right) \cdot f(\mathcal{Z})\, d\mathcal{Z} && (10.20)\\
&\leq \left(1 - e^{-2\lambda(\gamma+5\epsilon_l+\epsilon_u)}\right) \cdot e^{2\lambda\epsilon_u} \cdot \int_{\mathcal{Z}} \left(\mathbf{Pr}[\delta \geq \rho + \epsilon_u] + \frac{1}{e^{\lambda}-1}\right) \cdot f(\mathcal{Z})\, d\mathcal{Z} && (10.21)\\
&\leq \left(1 - e^{-2(\gamma+5\epsilon+\epsilon_u)\lambda}\right) \cdot e^{2\epsilon_u\lambda} \cdot \int_{\mathcal{Z}} \left(\mathbf{Pr}[\mathcal{F} \mid \mathcal{Z}] + \frac{1}{e^{\lambda}-1}\right) \cdot f(\mathcal{Z})\, d\mathcal{Z} && (10.22)\\
&= \left(1 - e^{-2(\gamma+5\epsilon+\epsilon_u)\lambda}\right) \cdot e^{2\epsilon_u\lambda} \cdot \left(\mathbf{Pr}[\mathcal{F}] + \frac{1}{e^{\lambda}-1}\right) && (10.23)
\end{aligned}$$

This proves the lemma. □

Using this lemma, we see that it holds:

$$\begin{aligned}
\sum_{i=1}^{|N_v|} \mathbf{Pr}[\mathcal{C}_i] &\leq \left(1 - e^{-2(\gamma+5\epsilon_l+\epsilon_u)\lambda}\right) \cdot e^{2\epsilon_u\lambda} \cdot \sum_{i=1}^{|N_v|} \left(\mathbf{Pr}[\mathcal{F}_i] + \frac{1}{e^{\lambda}-1}\right)\\
&\leq \left(1 - e^{-2(\gamma+5\epsilon_l+\epsilon_u)\lambda}\right) \cdot e^{2\epsilon_u\lambda} \cdot \left(1 + \frac{\tau}{e^{\lambda}-1}\right) && \text{As} \sum \mathbf{Pr}[\mathcal{F}_i] = 1
\end{aligned}$$

We can finalize the proof by carefully considering the possible values of γ, ϵ_l and ϵ_u. As $\epsilon_u = 1/n^c$ with $c > 1$ and $\lambda = 2\log\tau + 2 \leq 2\log n + 2$, we can assume $2\epsilon_u\lambda < 1/40\lambda \leq 1.79$ for a large enough n. This allows us to apply the well-known inequality $e^x \leq 1 + x + x^2$ for $x \leq 1.79$ and get

$$e^{2\epsilon_u\lambda} \leq 1 + 2\epsilon\lambda + (2\epsilon\lambda)^2 \leq 1 + 4\epsilon\lambda \tag{10.24}$$

Further, we picked $\gamma \leq 1/8$ and $\epsilon \leq 1/40$ small enough such that $\gamma + 5\epsilon_l + \epsilon_u \leq 1/4$. Given that observation, we see that it holds:

$$\begin{aligned}
e^{-2(\gamma+5\epsilon_l+\epsilon_u)\lambda} &= \frac{e^{-2(\gamma+5\epsilon_l+\epsilon_u)\lambda}\left(e^{\lambda}-1\right)}{e^{\lambda}-1} \geq \frac{e^{-2(\gamma+5\epsilon_l+\epsilon_u)\lambda} \cdot e^{\lambda-1}}{e^{\lambda}-1}\\
&\geq \frac{e^{-2(1/4)\lambda} \cdot e^{\lambda-1}}{e^{\lambda}-1} \geq \frac{e^{\frac{\lambda}{2}-1}}{e^{\lambda}-1}
\end{aligned}$$

Here, we used the fact that $e^{x-1} \leq e^x - 1$ for $x > 2$, which can be easily verified. By our choice of $\lambda := 2\log(\tau) + 2$, it furthermore holds:

$$e^{-2(\gamma+5\epsilon_l+\epsilon_u)\lambda} \geq \frac{e^{\frac{\lambda}{2}-1}}{e^{\lambda}-1} \geq \frac{e^{\frac{2\log(\tau)+2}{2}-1}}{e^{\lambda}-1} = \frac{e^{\log\tau}}{e^{\lambda}-1} \geq \frac{\tau}{e^{\lambda}-1} \tag{10.25}$$

Therefore, we can simplify our formula as follows:

$$\sum_{i=1}^{|N_v|} \mathbf{Pr}[\mathcal{C}_i] \leq \left(1 - e^{-2(\gamma+5\epsilon_l+\epsilon_u)\lambda}\right) \cdot e^{2\epsilon_u \lambda} \cdot \left(1 + \frac{\tau}{e^{\lambda}-1}\right)$$

By Ineq. (10.24) :

$$\leq (1+4\lambda\epsilon_u) \cdot \left(1 - e^{-2(\gamma+5\epsilon_l+\epsilon_u)\lambda}\right)\left(1 + \frac{\tau}{e^{\lambda}-1}\right)$$

By Ineq.(10.25) :

$$\leq (1+4\lambda\epsilon_u) \cdot \left(1 - e^{-2(\gamma+5\epsilon_l+\epsilon_u)\lambda}\right)\left(1 + e^{-2(\gamma+5\epsilon_l+\epsilon_u)\cdot\lambda}\right)$$

As $(1+x)(1-x) = 1^2 - x^2$:

$$\begin{aligned}
&= (1+4\lambda\epsilon_u)\left(1 - e^{-4(\gamma+5\epsilon_l+\epsilon_u)\cdot\lambda}\right)\\
&\leq 1 - e^{-4(\gamma+5\epsilon_l+\epsilon_u)\cdot\lambda} + 4\lambda\epsilon_u\\
&\leq 1 - e^{-4(\gamma+5\epsilon+5\epsilon_u+\epsilon_u)\cdot\lambda} + 4\lambda\epsilon_u\\
&= 1 - e^{-4(\gamma+5\epsilon)\cdot\lambda} \cdot e^{-24\epsilon_u\cdot\lambda} + 4\lambda\epsilon_u
\end{aligned}$$

As $e^{-x} \leq 1 - x/2$:

$$\begin{aligned}
&\leq 1 - e^{-4(\gamma+5\epsilon)\cdot\lambda} \cdot (1 - 12\epsilon_u \cdot \lambda) + 4\lambda\epsilon_u\\
&\leq 1 - e^{-4(\gamma+5\epsilon)\cdot\lambda} + 12\epsilon_u \cdot \lambda + 4\lambda\epsilon_u\\
&\leq 1 - e^{-4(\gamma+5\epsilon)\cdot\lambda} + 16\epsilon_u \cdot \lambda
\end{aligned}$$

Putting this bound on $\sum_{i=1}^{|N_v|} \mathbf{Pr}[\mathcal{C}_i]$ back in our initial inequality gives us the following final approximation of our desired probability:

$$\begin{aligned}
\mathbf{Pr}[\exists x_i \in N_v : \Upsilon_i \leq -(2\gamma+10\epsilon)\cdot\mathcal{D}] &\geq 1 - \sum_{i=1}^{\tau} \mathbf{Pr}[\mathcal{C}_i]\\
&\geq 1 - \left(1 - e^{-4(\gamma+5\epsilon)\cdot\lambda} + 16\lambda\epsilon\right)\\
&= e^{-4(\gamma+5\epsilon)\cdot\lambda} + 16\lambda\epsilon_u
\end{aligned}$$

Recalling that $\lambda = 2\log\tau + 2 \leq 4\log\tau$:

$$\begin{aligned}
&\geq e^{-4(\gamma+5\epsilon)\cdot(4\log\tau)} - 16(4\log\tau)\epsilon_u\\
&\geq e^{-16(\gamma+5\epsilon)\cdot(\log\tau)} - 64\epsilon_u\log\tau\\
&= e^{-16(\gamma+5\epsilon)\cdot(\log\tau)} - \frac{64\log\tau}{n^c}
\end{aligned}$$

This proves Lemma 10.5.

10.3 Low-Diameter Clusterings from Pseudo-Padded Decompositions

In this section, we present an algorithm that creates an LDC of strong diameter $\mathcal{D}$ where each edge is cut with a probability proportional to its length. To achieve this, we will generalize an algorithm by Becker, Emek, and Lenzen [BEL20] that creates such a decomposition for general graphs. Our main result is a generic algorithm that creates an LDC with strong diameter $\mathcal{D}$ of quality $(O(\log \tau), 1/2)$ if the number of nodes that can be centers of clusters has been sufficiently sparsed out, such that each node can only be in one of τ clusters. We show that it holds:

Theorem 10. *A Generic Clustering Theorem*

Let $\mathcal{D} > 0$ be a distance parameter, $G := (V, E, \ell)$ a (possibly weighted) undirected graph, and $\mathcal{X} \subseteq V$ be a set of marked nodes. Suppose that for each node $v \in V$, the following two properties hold:

- **Covering Property:** There is at least one $x \in \mathcal{X}$ with $d(v, x) \leq \mathcal{D}$.
- **Packing Property:** There are at most τ centers $x' \in \mathcal{X}$ with $d(v, x') \leq 6\mathcal{D}$.

Then, there is an algorithm that creates an LDC of strong diameter $8\mathcal{D}$ with quality $\left(O\left(\log \tau / \mathcal{D}\right), 1/2\right)$.

The algorithm can be implemented with $\tilde{O}(1)$ minor aggregations and $\tilde{O}(1)$ $(1+\epsilon)$-approximate SetSSP computations where $\epsilon \in O\left(1/\log^2 n\right)$.

The algorithm is based on two main techniques: First, it makes use of the so-called *blurry ball growing* (BBG) technique. This is perhaps the key technical result that Becker, Emek, and Lenzen introduced in [BEL20]. It provides us with the following guarantees:

Lemma 10.7 (Blurry Ball Growing (BBG), cf. [BEL20, REGH22]). *Given a subset $S \subseteq V$ and an arbitrary parameter $\rho > 0$, an algorithm for BBG outputs a superset $S' \supseteq S$ with the following properties*

1. *An edge $\{v, w\} \in E$ of length $\ell_{(v,w)}$ is cut with probability*

$$\mathbf{Pr}[v \in S', w \notin S'] \leq O\left(\frac{\ell_{(v,w)}}{\rho}\right).$$

2. *For each node $v \in S'$, it holds $d(v, S) \leq \frac{\rho}{1-\alpha} \leq 2\rho$ where $\alpha \in O\left(\log\log n / \log n\right)$.*

BBG can be implemented using $\tilde{O}(1)$ $(1+\epsilon)$ approximate SetSSP computations with $\epsilon \in O\left(\left(\log\log n / \log n\right)^2\right)$.

This technique allows us to create clusters with a low probability of cutting an edge while only having access to approximate shortest paths. Note that in [REGH22] the dependency on ϵ was improved to $O\left(1/\log n\right)$. The paper also presents a deterministic version of blurry ball growing. However, we will only use the probabilistic version introduced above. Second, the algorithm uses the following technical fact about the pseudo-padded decompositions computed by Theorem 9.

Lemma 10.2 (Padding Property). *Consider node $v \in V$ clustered by center $x_v \in \mathcal{X}$ and a parameter $\epsilon \leq \gamma \leq \frac{1}{8}$. Further, let $P_{x_v} = (v_1, \dots, v_\ell)$ be the **exact** shortest path from x_v to v. Then, for each $v_i \in P_{x_v}$, the ball $B_G(v_i, \gamma\mathcal{D})$ is fully contained in C_{x_v} with probability at least*

$$\mathbf{Pr}\left[\bigcup_{v_i \in P_{x_v}} B_G(v_i, \gamma\mathcal{D}) \subset C_{x_v}\right] \geq e^{-16(\gamma+5\epsilon)\log\tau} - \frac{160\log\tau}{n^c}$$

This lemma states that if a single node $v \in V$ is padded, all nodes on the shortest path to its cluster center are likely padded as well. While this initially sounds very technical, it roughly translates to the following: Consider a clustering $\mathcal{K} = K_1, \dots, K_N$. Then for a suitably chosen $\gamma' \in \Theta(\log\tau)$, in each cluster $\mathcal{K}_i = (V_i, E_i)$ there is a constant fraction of nodes $V_i' \subseteq V$ in the distance $\gamma'\mathcal{D}$ to their closest node in a neighboring cluster $K_j \neq K_i$ **and** for any two $v, w \in V_i'$ there is a path of length $4\mathcal{D}$ (via the cluster center) that only consists of nodes in V_i'.

Given these two preliminaries, we now give an overview of the algorithm that creates an LDC. For brevity, we omit the exact values of certain parameters. The algorithm first computes a pseudo-padded decomposition $\mathcal{K} = K_1, \dots, K_N$ of strong diameter $\mathcal{D}$ using the algorithm of Theorem 9. We choose the set $\mathcal{X}$ as the set of cluster centers. Within each cluster K_i, we then determine an *inner cluster* $K_i' \subseteq K_i$ of strong diameter $3\mathcal{D}$ where each node $v \in V_i'$ has distance $\rho' \in O(\frac{D}{\log\tau})$ to the closest node in different cluster $K_j \neq K_i$. These inner clusters can be determined via two (approximate) SetSSP computations: First, all nodes calculate the 2-approximate distance to the closest node in a different cluster. We call all nodes where this 2-approximate distance exceeds $2\rho'$ *active nodes*. Then, the active nodes compute if they have a path of length at most $3\mathcal{D}$ to their cluster center using only active nodes. If so, they add themselves to the inner cluster K_i'. Their exact distance to the next cluster is at least ρ' as the paths are 2-approximate. Further, for any two nodes $v, w \in K_i'$ there is a path of length $6\mathcal{D}$ via the cluster center. Thus, this procedure *always* results in an inner cluster K_i' with the desired properties. We can prove that the inner clusters are large using Lemma 10.2. For a constant fraction of the nodes $v \in V$ the following holds: Node $v \in K_i$ **and** all nodes on the path to its cluster center are distance at least $2\rho'$ to the closest node in a different cluster. Therefore, with a probability of at least $1/2$, node v and all nodes on the path to the cluster center will be active. As the cluster center is distance $2\mathcal{D}$, they all will be added to the inner cluster. Thus, these inner clusters will contain half of all nodes in expectation.

Then, the *inner clusters* are refined using blurry ball growing (BBG) to obtain the correct probability of cutting an edge.

We choose the parameter for the BBG to be $\rho \in O(\mathcal{D}/\log n)$. In particular, choose $2\rho < \rho'$. As the distance to the next cluster is ρ' and BBG only adds nodes in distance $2\rho < \rho'$, for each inner cluster K_i', this produces a well-defined super-set $S(K_i')$. We pick these supersets as our low-diamter clustering. The choice of ρ implies that edges are cut with correct probability of $O(\rho \cdot \ell) = O(\frac{\ell \cdot \log\tau}{\mathcal{D}})$ while their diameter is at most $6\mathcal{D} = 4\rho \leq 8\mathcal{D}$. Theorem 10 follows as half of all nodes are in an inner cluster.

We describe the algorithm in more detail in Section 10.3.1. There, we will provide the exact choice of parameters we use to find the inner clusters and to execute the BBG. In Section 10.3.2, we analyze the algorithm and prove Theorem 10.

10.3.1 Algorithm Description

We now present the algorithm behind Theorem 10 in more detail. As input, we are given a weighted graph $G = (V, E, \ell)$ with polynomially bounded weights, a diameter bound $\mathcal{D}$, and the centers $\mathcal{X} \subseteq V$. On a high level, the algorithm consists of four stages: In the first stage, we compute an initial decomposition by the algorithm from Theorem 9. We want to use the blurry ball growing from Lemma 10.7 on these clusters to get the desired cut probabilities. However, we cannot directly use thems as input sets because the resulting balls would overlap. Instead, we have to ensure somehow that the input sets are separated from each other. To this end, we shrink the clusters to avoid overlaps. This works in two steps: First, by identifying all nodes in sufficient distance to other clusters, and then shrinking the clusters in a specific way (which differs for the two the diameter guarantees). Finally, we get the desired decomposition by using a blurry ball growing on each shrunk cluster. As a parameter for the blurry ball growing, we choose

$$\rho := (1-\alpha) \cdot \left(\frac{\mathcal{D}}{c_{\mathsf{blur}} \log(\tau)}\right) \tag{10.26}$$

Here, $\alpha \in O\left(\log\log n / \log n\right)$ is the value from Lemma 10.7 and c_{blur} is a (large) constant that will be determined in the analysis. Further, we fix

$$\epsilon := \frac{1}{1024 \cdot \log \tau} \tag{10.27}$$

as the parameter for approximate SetSSP computations. In more detail, the four stages are as follows:

(Step 1) Compute a Pseudo-Padded Decomposition: First, we create a pseudo-padded decomposition using the algorithm from Theorem 9.

(Step 2) Compute (Approximate) Distance to Boundary: Each node computes the approximate distance to the closest node in a different cluster. This works in two substeps

- **(Substep 2a) Identify Nodes on Boundary:** All the nodes of a cluster that are on the boundary of that cluster, i.e., nodes that are adjacent to another cluster, compute their (approximate) distance to the neighboring cluster. A node can determine whether it is on the boundary by checking its neighbors. Let $v \in V$ be a node on the boundary and let N'_v be a subset of its neighbors, s.t., each $w \in N'_v$ is a neighbor in a different cluster. Then, node v computes:

$$\omega(v) = \min_{w \in N'_v} d(v, w) \tag{10.28}$$

- **(Substep 2b) Compute (Approximate) Distance to Boundary:** Finally, we create another virtual source s' and all nodes v on the boundary create an edge of length $\omega(v)$. Then perform an $(1+\epsilon)$-approximate shortest path from s' and let each node compute $d_T(s', v)$.

(Step 3) Identify Well-Separated Nodes: We identify *connected* sets of nodes with large distance to all other connected sets. This works in two substeps:

- **(Substep 3a) Find Nodes With Sufficient Distance to Boundary:** Each nodes examines the distance $d_T(s', v)$ computed in the previous step. Suppose it holds

$$d_T(s', v) \geq \left(\frac{\rho}{1-\alpha} + \epsilon \mathcal{D}\right)$$

  we mark v as *active* and set

$$V_a := \left\{ v \in V \mid d_T(v, s') \geq \left(\frac{\rho}{1-\alpha} + \epsilon \mathcal{D}\right)\right\}$$

- **(Substep 3b) Find Connected Set of Active Nodes:** Let $G[V_a]$ be the graph induced by the active nodes. A node can locally decide which edges are part of this graph by checking which neighbors are active. Let $\mathcal{X}_a = \mathcal{X} \cap V_a$ be the set of active centers. Perform $(1+\epsilon)$-approximate SetSSP from $\mathcal{X}_a$ in $G[V_a]$. Let T_a be the resulting approximate SSSP tree. Then, if and only if an active node has a path of length $3\mathcal{D}$ to its cluster center in T_a, it remains active. We denote the remaining active nodes as V_a'.

(Step 4) Execute Blurry Ball Growing: We execute a blurry ball growing on the active set V_a' with parameter $\rho = (1-\alpha) \cdot \left(\frac{\mathcal{D}}{c_{\text{blur}} \log(\tau)}\right)$. As per Lemma 10.7, we obtain a superset $S(V_a')$ of V_a'.

(Step 5) Create a Spanning Tree: We perform one last approximate set-source SSSP from all centers $x \in \mathcal{X} \cap V_a'$. Each node $v \in V_a'$ adds itself to the cluster of the closest center. We add all these the refined clusters to our decomposition.

Note that the algorithm of Becker, Emek, and Lenzen only differs in Substep (3b), which is simply absent in their algorithm. Instead they perform BBG on the active nodes computed in in Substep (3a). However, this is exactly the additional step which ensures that the resulting clusters are connected and therefore have a strong diameter.

10.3.2 Analysis

We now prove Theorem 10. The proof is divided into three parts. First, prove that the cluster's diameter is bounded by $8\mathcal{D}$ in Lemma 10.8. After that, we analyze the cutting probabilities in Lemma 10.9 and show that each edge is cut with probability at most $O\left(\frac{\alpha\ell}{\mathcal{D}}\right)$. Finally, we show the algorithm's complexity in Lemma 10.12. Together, Lemmas 10.8, 10.9, and 10.12 prove Theorem 10.

Cluster Diameter: To start off the analysis, we show that the algorithm creates a clustering of diameter $8\mathcal{D}$. This follows more of less directly from the construction.

Lemma 10.8 (Diameter Guarantee). *Each cluster created in the last step has strong diameter at most $8\mathcal{D}$.*

First, we note the diameter guarantee follows **if** the blurry ball growing process never adds two nodes from different clusters to the same ball. To prove this, let A_i be the connected set of active nodes in the cluster of x_i. Note that we enforce that all active nodes have a path of length $3\mathcal{D}$ to a center that is completely contained in the cluster. Thus, the claim already follows for the nodes in A_i. Further, recall that for all nodes w that are added to the cluster through the blurry ball growing, it holds:

$$d(w, A_i) = d(w, V_a') \leq \frac{\rho}{1-\alpha} = \frac{(1-\alpha) \cdot \left(\frac{\mathcal{D}}{c_{\text{blur}} \log(\tau)}\right)}{1-\alpha} = \left(\frac{\mathcal{D}}{c_{\text{blur}} \log(\tau)}\right) \leq \mathcal{D} \tag{10.29}$$

Now consider $v, w \in V$ to be two nodes added to cluster K_i of center $x_i \in \mathcal{X}$. Then, it holds:

$$\begin{aligned} d_{K_i}(v, w) &\leq d_G(v, A_i) + \max_{v', w' \in A_i} d_G(v, V_a') + d_G(v, A_i) &(10.30)\\ &\leq \mathcal{D} + d_G(v', x_i) + d_G(w', x_i) + \mathcal{D} &(10.31)\\ &\leq 2(3\mathcal{D} + \mathcal{D}) = 8\mathcal{D}. &(10.32)\end{aligned}$$

Thus, to prove the diameter guarantee, we must argue why we can safely execute blurry balls growing from the active nodes without adding active nodes of a different cluster.

Claim 16. *For each active node $v \in V_a$, all nodes in distance $\frac{\rho}{1-\alpha}$ are part of the same cluster.*

Proof. For contradiction, assume there is node $u \in V$ distance smaller than $\frac{\rho}{1-\alpha}$ to v that is in a different cluster. In particular, let u be the closest such node. Clearly, if we prove a contradiction for u, it holds for all other nodes, too.

We begin with the following observation: As there can be no other node in a different cluster on the exact shortest path from u to v in G (as this node would closer to v than u), there must be boundary node $w \in V$, s.t., the following two conditions hold:

1. w is on the path from u to v. Therefore, it holds $d(u, v) = d(u, w) + d(w, v)$.

2. w adds a virtual edge of length $\omega(w) \leq d(u, w)$ to the virtual source s.

Together, these two facts imply:

$$d_{G_s}(s, v) \leq d_G(u, v) \tag{10.33}$$

Here, G_s is the graph we obtain by adding the (virtual) supernode to G. Therefore, it holds:

$$\begin{aligned} \frac{\rho}{1-\alpha} &> d_G(u, v) && \triangleright \textit{Per Assumption}\\ &\geq d_{G_s}(s, v) && \triangleright \textit{By Ineq. } (10.33)\end{aligned}$$

Now recall that v is an active node. Otherwise, it would not be part of the set that is blurred. Thus, it holds

$$d_T(s,v) \geq \frac{\rho}{1-\alpha} + \epsilon\mathcal{D} \tag{10.34}$$

by the definition of active nodes. Therefore, it holds

$$\begin{aligned}
d_{G_s}(s,v) &\geq \frac{d_T(s,v)}{1+\epsilon} && \triangleright \textit{As } d_T(s,v) \leq (1+\epsilon)d_{G_s}(s,v) \\
&\geq \frac{\left(\frac{\rho}{1-\alpha} + \epsilon\mathcal{D}\right)}{1+\epsilon} && \triangleright \textit{As } v \textit{ is active} \\
&\geq \frac{(1+\epsilon)\frac{\rho}{1-\alpha}}{1+\epsilon} && \triangleright \textit{As } \frac{\rho}{1-\alpha} \leq \mathcal{D} \\
&\geq \frac{\rho}{1-\alpha}
\end{aligned}$$

This is a contradiction as this implies:

$$\frac{\rho}{1-\alpha} > d_{G_s}(s,v) \geq \frac{\rho}{1-\alpha}$$

So, node u cannot exist, which proves the lemma. □

Therefore, it is safe to execute the ball growing on a subset of active nodes without risking overlapping with other clusters. Thus, all balls must have a diameter of $8\mathcal{D}$

Edge-Cutting Probability Next, we consider the probability of cutting edges. Altogther, we want to show that the following holds:

Lemma 10.9. *An edge of length ℓ is cut with probability at most $O\left(\frac{\ell \cdot \log \tau}{\mathcal{D}}\right)$.*

Recall that the only operation that cuts edges is the BBG procedure, and so at first glance, Lemma 10.7 should immediately yield the result. However, this only holds true for a single iteration. The main difficulty of this proof is that the algorithm runs multiple iterations until all nodes are clustered, and therefore, the BBG has many *chances* to cut a given edge. To handle this, we will exploit that each edge is cut or added to a cluster after at most 2 iterations on expectation.

More precisely, the proof is structured as follows: First, we consider only a single iteration and show that each edge is cut with probability at most $O\left(\frac{\ell \cdot \log \tau}{\mathcal{D}}\right)$. Further, we show that a node is added to a cluster with constant probability, and so all its edges must be gone after $O(\log n)$ iterations, w.h.p. Then, we can use a (more or less) standard argument from the study of LDD's. It roughly goes as follows: Any clustering algorithm $\mathcal{A}$ that cuts edges between cluster with some probability α and places a constant fraction of nodes into clusters, can be turned into an LDD with quality $O(a)$. The idea is simple and straightforward. Recursively apply the algorithm until all nodes are clustered. For the sake of completeness, we prove this general statement and use it finish our analysis.

We begin with the cutting probability of a single iteration. Here, it holds:

Lemma 10.10. ***In a fixed iteration**, an edge of length ℓ is cut with probability at most $O\left(\frac{\ell \cdot \log \tau}{\mathcal{D}}\right)$.*

Proof. Note that the blurry ball growing is the only operation that removes edges. Following Lemma 10.7, the probability that an edge is cut by this process is $\ell \cdot \rho$. By our choice of ρ, the short edges are cut with probability

$$\begin{aligned}\frac{\ell}{\rho} = \ell \cdot \left(\frac{\mathcal{D}}{c_{\mathsf{blur}} \log \tau} (1-\alpha)\right)^{-1} &:= \ell \cdot \left(\frac{\mathcal{D}}{c_{\mathsf{blur}} \log \tau}\left(1 - O\left(\frac{\log\log n}{\log n}\right)\right)\right)^{-1} \\ \leq \ell \cdot \left(\frac{\mathcal{D}}{c_{\mathsf{blur}} \log \tau}\left(1 - \frac{1}{2}\right)\right)^{-1} &\leq \ell \cdot \left(\frac{\mathcal{D}}{2c_{\mathsf{blur}} \log \tau}\right)^{-1} \in O\left(\frac{\ell \cdot \log \tau}{\mathcal{D}}\right)\end{aligned}$$

This proves the desired bound on the probability. □

Clustering Probability Further, we lower bound how many nodes are actually added to a cluster. Here, the analysis differs from Becker, Lenzen, and Emek and we use our tighter probability bounds on the behaviour of the Pseudo-Padded Decomposition. Whereas they show that an *individual* node is active with constant probability in Substep (3a), we show that this also implies that (with constant probability) all nodes on the path its cluster center are active as well. For this, we use a technical observation from the analysis of Theorem 9. We show that the following holds:

Lemma 10.11 (Clustering Probability). *In a fixed iteration, each node $v \in V$ is clustered with probability at least $\beta \geq 1/2$.*

Proof. Recall that each *active* node in the set V_a' is added to a cluster no matter what. To this end, we will analyse how many nodes are active. For the analysis, define $\gamma = (\frac{1}{c_{\mathsf{blur}} \log \tau} + \epsilon)$. Further, let x_v now be the center clusters v in the pseudo-padded decomposition. Note that a node $v \in V$ is definitely active if the ball $B(v, \gamma\mathcal{D})$ is fully contained in the same cluster as v. In this case, the boundary is in distance at least $\gamma\mathcal{D}$. Recall our approximate shortest path algorithm only *overestimates* and therefore v passes the check and is added to V_a.

This alone is not sufficient to be also added to V_a'. We must additionally show that a path of length at most $\frac{3\mathcal{D}}{(1+\epsilon)}$ to x_v also remains active. To this end, let $P_{x_v} = (v_1, \dots, v_\ell)$ be the **exact** shortest path from x_v to v in G. We will show that with constant probability, *all* nodes on this path (including x_v) are active. Note that this path is at most of length $(1+\epsilon)2\mathcal{D}$ as per 13 from the analysis in the previous section. This implies that there is path of active nodes of length $(1+\epsilon)2\mathcal{D}$. For small enough ϵ, it holds $(1+\epsilon)2\mathcal{D} \leq \frac{3\mathcal{D}}{(1+\epsilon)}$. Thus, our approximate shortest path algorithm must find a path of length at most $(1+\epsilon)\frac{3\mathcal{D}}{(1+\epsilon)} \leq 3\mathcal{D}$ in the induced active graph $G[V_a]$. Therefore, v will remain active, is added to V_a', and therefore will be in a cluster.

Now, recall the analysis of Theorem 9. There, we proved that:

Lemma 10.2 (Padding Property). *Consider node $v \in V$ clustered by center $x_v \in \mathcal{X}$ and a parameter $\epsilon \leq \gamma \leq \frac{1}{8}$. Further, let $P_{x_v} = (v_1, \dots, v_\ell)$ be the **exact** shortest path from x_v to v. Then, for each $v_i \in P_{x_v}$, the ball $B_G(v_i, \gamma\mathcal{D})$ is fully contained in C_{x_v} with probability at least*

$$\mathbf{Pr}\left[\bigcup_{v_i \in P_{x_v}} B_G(v_i, \gamma\mathcal{D}) \subset C_{x_v}\right] \geq e^{-16(\gamma+5\epsilon)\log\tau} - \frac{160\log\tau}{n^c}$$

This allows us to prove the lemma. Recall $c_{\text{blur}} \geq 128$ and we execute a pseudo padded decomposition with padding parameter $\lambda = 2\log \tau + 2$, diameter parameter $\mathcal{D}$, and $\epsilon \leq \frac{1}{1024 \log \tau}$. Let $\gamma = (\frac{1}{c_{\text{blur}} \log \tau} + \epsilon)$. Note that $\gamma \leq \frac{1}{8}$ for a our choice of $1/c_{\text{blur}} \leq \frac{1}{128}$ and $\epsilon \leq \frac{1}{1024 \log \tau}$. Finally, define:

$$\mathcal{F}_v := \bigcup_{w \in P_{x_v}} \{B_G(w, \gamma\mathcal{D}) \subset C_{x_v}\}$$

Then, by Lemma 10.2, it holds:

$$\begin{aligned}
\mathbf{Pr}[\mathcal{F}_v] &\geq e^{\left(-16 \cdot \log \tau \cdot \left(\frac{1}{c'_{\text{blur}} \log \tau} + 6\epsilon\right)\right)} - \frac{160\log \tau}{n^c} \\
\text{Using that } e^x \geq 1 + x: & \\
&\geq 1 - 16 \cdot \log \tau \cdot \left(\frac{1}{c'_{\text{blur}} \log \tau} + 6\epsilon\right) - \frac{160\log \tau}{n^c} \\
\text{Using that } 1/n^c \leq \epsilon: & \\
&= 1 - \frac{16 \log \tau}{c_{\text{blur}} \log \tau} - 256 \cdot \epsilon\log \tau \\
&= 1 - \frac{16}{c_{\text{blur}}} - 256 \cdot \epsilon\log \tau \\
\text{Using that } c_{\text{blur}} \geq 128 = 4 \cdot 16: & \\
&\geq 1 - \frac{1}{4} - 256 \cdot \epsilon\log \tau \\
\text{Using that } \epsilon \leq 1/1024 \log \tau = 1/4 \cdot 256 \cdot \log \tau: & \\
&\geq 1 - \frac{1}{4} - \frac{1}{4} = \frac{1}{2}
\end{aligned}$$

Therefore, with constant probability, a node is active all nodes on the shortest path to its cluster center are also active. This means, there is a path of length $3\mathcal{D}$ to the cluster center that only consists of active nodes. Thus $v \in V'_a$ with probability at least $1/2$ as needed. □

Complexity Finally, we consider the algorithm's runtime. The follows directly from the algorithm description as it only uses minor aggregations and approximate shortest-path computations.

Lemma 10.12 (Complexity). *The algorithm can be implemented with $\tilde{O}(1)$ approximate SetSSP computations and minor aggregations, w.h.p.*

Proof. As there can be at mots $O(\log n)$ iterations, w.h.p, we focus on a single iteration. Each iteration begins with one execution of the pseudo-padded decomposition; this requires only SetSSP computations with approximation parameter $O(1/\log n)$. This follows directly from Theorem 9.

Next, we require one aggregation to let the boundary nodes compute the distances to nodes in neighboring clusters. To be precise, each node $v \in V$ is its own minor. We perform no aggregation within the minor and jump directly to the last step where we perform a aggregation on the edges. In other words, we skip the consensus step and directly proceed to the aggragation step. For each incident edge $\{v, w\} \in E$, node $v \in V$ chooses the input $(x_v, \ell_{(v,w)})$. Here, $x_v \in \mathcal{X}$ is the (identifier of the) center that clustered v. Then, we aggregate the minimum of all values with $x_w \neq x_v$ (by considering inputs containing x_v as ∞). Thus, each node learns the distance closest *adjacent* node in a different cluster with *one* minor aggregation. To conclude the second step, we perform one approximate SSSP on graph $G_{s'}$ that contains one virtual node s'. The approximation parameter is $\epsilon \in O(1/\log n)$.

In the third step, we need another local aggregation to determine which nodes are active. Similar to the step before, each node performs an aggregation on all its edges. Then, we perform an approximate SSSP on the graph induced by the active nodes. The approximation parameter is again $\epsilon \in O(1/\log n)$ Finally, we perform one execution of the blurry ball growing, which only uses $\tilde{O}(1)$ approximate SetSSP computations with parameter $\alpha \in O(\log\log n/\log n)$ as per Lemma 10.7. A final SSSP to determine the connected components conludes the step. Thus, each step only performs $\tilde{O}(1)$ approximate SetSSP computations or aggregations. In all cases, the approximation factor is within $\Omega(1/\log^2)$. Altogether, this proves the runtime bounds from Theorem 10. $\square$

10.4 Low-Diameter Decompositions for General Graphs

In this section, we want to prove the first statement from Theorem 8. More precisely, we want to show that we can create an LDD with quality $O(\log n)$ for any graph within $\tilde{O}(1)$ minor aggregations and $(1+\epsilon)$-approximate SetSSP's. We show that:

Theorem 11. *A Clustering Theorem for General Graphs*

Let $\mathcal{D} > 0$ be an arbitrary distance parameter and $G := (V, E, \ell)$ be a (possibly weighted) undirected graph. Then, there is an algorithm that creates an LDD of G with strong diameter $\mathcal{D}$ and quality $O(\log n)$.

Note that, at first glance, the theorem follows almost directly from the Theorem. To see it, choose $\mathcal{X} = V$ and apply the theorem to a graph G. By choosing $\mathcal{D}' = 8\mathcal{D}$, we obtain a clustering with diameter $\mathcal{D}'$ if each node has one node in distance $\mathcal{D}'/8$ and at most τ nodes in distance $\mathcal{D}'$. Clearly, for every possible choice of parameter $\mathcal{D}'$, each node has at least one marked node in distance $\mathcal{D}'/8$, namely itself. Further, for every possible choice of parameter $\mathcal{D}'$, each node has at most n marked node in distance $\mathcal{D}$ because there are only n nodes in total. Thus, by choosing all nodes as centers, we obtain a clustering with quality $O(\log n)$ and a diameter we can choose freely.

While the diameter and edge-cutting probability of the resulting clustering are (asymptotically) correct, we can only guarantee that at least half of the nodes are clustered on expectation. To put it simply, we do not have a partition as required. However, we can reapply our algorithm to the graph implied by unclustered nodes. This clusters half of the remaining nodes on expectation. It is easy to see that, if we continue this for $O(\log n)$ iterations, we obtain the desired partition, w.h.p. Further, this will not significantly affect the edge-cutting probability, as each edge that is not cut will be added to a cluster with constant probability. Therefore, there cannot be too many iterations where the edge can actually be cut. Thus, applying the algorithm until all nodes are clustered intuitively produces the required LDD of quality $O(\log n)$. We formalize and prove this intuition in the following lemma. Not that we state it in a more general fashion to reuse later

Lemma 10.13. *Let $G = (V, E, \ell)$ be a weighted graph. Let $\mathcal{A}$ be an algorithm that for each subgraph of $G' \subseteq G$ can create clusters $K_1, \dots, K_N$ where each edge is cut between clusters with probability at most α, and each node is added to a cluster with probability at least β. Then, recursively applying $\mathcal{A}$ to G until all its nodes are in a cluster creates an LDD of quality $O(\alpha\beta^{-2})$.*

The procedure requires $O(\beta^{-1}\log n)$ applications of $\mathcal{A}$, w.h.p.

Proof. Let $G_0 := G$ and define $G_i := (V_i, E_i)$ with $i \geq 1$ the remaining graph after we apply the algorithm $\mathcal{A}$ on G_{i-1}. On expectation, in each application of $\mathcal{A}$, a β-fraction of the nodes is added to some cluster. This follows from the definition of β and the linearity of expectation: For each node $v \in V$, let X_v^i be the binary random variable that v gets clustered in the i^{th} application of the algorithm, i.e., whether v will be in G_{i+1} or not. It holds $\mathbf{Pr}[X_i \mid v \in V_i] \geq \beta$ by definition. By the linearity of expectation, it therefore holds:

$$\begin{aligned}
\mathbb{E}\left[|V_{i+1}|\right] &= \sum_{v \in V} \mathbf{Pr}[v \in V_i]\left(1 - \mathbf{Pr}[X_i \mid v \in V_i]\right) \\
&\leq (1-\beta) \sum_{v \in V} \mathbf{Pr}[v \in V_i] \\
&= (1-\beta) \cdot \mathbb{E}\left[|V_i|\right]
\end{aligned}$$

Thus, a simple induction reveals that after t applications, it holds:

$$\mathbb{E}\left[|V_t|\right] \leq \mathbb{E}\left[|V_t|\right] \cdot (1-\beta)^t = n \cdot (1-\beta)^t$$

After $t = (c+1) \cdot \beta \cdot \log n$ recursive steps, the expected number of nodes is less than

$$\begin{aligned}
\mathbb{E}\left[|V_t|\right] &\leq n \cdot (1-\beta)^t \\
&= n \cdot (1-\beta)^{(c+1)\cdot\beta\cdot\log n} \\
&\leq n \cdot e^{-(c+1)\log n} = n^{-c}
\end{aligned}$$

Here, we used that $(1 - \frac{1}{x})^x \approx \frac{1}{e}$. Therefore, using Markov's inequality, we have

$$\begin{aligned}
\mathbf{Pr}[|V_t| \geq 1] &= \mathbf{Pr}\left[|V_t| \geq n^{-c} \cdot n^c\right] \\
&\leq \mathbf{Pr}[|V_t| \geq n^c \cdot \mathbb{E}\left[|V_t|\right]] \\
&\leq {}^{1}/_{n^c}
\end{aligned}$$

Thus, there are no nodes left, w.h.p, proves the runtime of $O(\beta^{-1}\log n)$ application of $\mathcal{A}$.

Next, we consider the cutting probability. Fix an edge $z \in E$ and denote by Cut_z the event that this edge is cut in any of the $O(\beta^{-1}\log n)$ iterations. We need to show that

$$\mathbf{Pr}[\mathsf{Cut}_z] \leq O\left(\frac{\alpha \cdot \ell_z}{\beta^2 \cdot \mathcal{D}}\right).$$

Recall that *being cut* means that one endpoint of z is added to a cluster, but the other is added to a different cluster (or not at all). In the following, we say that an edge gets *decided* if either endpoint is added to some cluster. If this happens, the other endpoint is added to the same cluster or not. In the former case, the edge is saved as it cannot be cut in any future iteration. In the latter case, the edge is irreversibly cut. For a fixed edge $e \in E$, let Y_i be the event that the edge is decided in recursive step i. Recall that either endpoint of an edge $z \in E_i$ that is still fully contained in the graph is clustered with probability at least β. Thus, the probability that *any* of the two endpoints is added to some cluster is also at least β. Therefore, the probability that no endpoint is added to a cluster in the first $i-1$ iterations is at most:

$$\mathbf{Pr}[Y_i] \leq (1-\beta)^{i-1} \tag{10.35}$$

Conditioned on the event Y_i, the edge can *only* be cut in the i^{th} iteration. In particular, the probability of a cut is 0 in the first $i-1$ iterations, as the event Y_i implies that both endpoints survive these iterations. Further, it cannot be cut later as it will not exist in subsequent graphs. Seeking formalization of these two observations, let now Cut_z^i be the indicator that z is cut in the i^{th} iteration. Then, the probability that an edge is cut **under the condition that one endpoint is added to a cluster in G_i** is:

$$\begin{aligned}
\mathbf{Pr}[\mathsf{Cut} \mid Y_i] &= \mathbf{Pr}[\mathsf{Cut} \mid \{z \in E_i\} \wedge \{z \notin E_{i+1}\}] \\
&= \lim_{T\to\infty} \sum_{i=1}^{T} \mathbf{Pr}\left[\mathsf{Cut}_z^i \mid \{z \in E_i\} \wedge \{z \notin E_{i+1}\}\right] \\
&= \mathbf{Pr}\left[\mathsf{Cut}_z^i \mid \{z \in E_i\} \wedge \{z \notin E_{i+1}\}\right]
\end{aligned}$$

Recall that, in a fixed iteration, the edge z is cut with probability at most $O\left(\frac{\ell_z \cdot \alpha}{\mathcal{D}}\right)$. However, we need to condition this probability on the fact that the edge gets decided in the i^{th} iteration. Using the law of total probability, we see:

$$\begin{aligned}
\mathbf{Pr}\left[\mathsf{Cut}_z^i \mid z \in E_i\right] =& \mathbf{Pr}[\{z \notin E_{i+1} \mid z \in E_i] \cdot \mathbf{Pr}\left[\mathsf{Cut}_z^i \mid \{z \in E_i\} \wedge \{z \notin E_{i+1}\}\right] \\
&+ \mathbf{Pr}[\{z \in E_{i+1} \mid z \in E_i] \cdot \mathbf{Pr}\left[\mathsf{Cut}_z^i \mid \{z \in E_i\} \wedge \{z \in E_{i+1}\}\right] \\
=& \mathbf{Pr}[\{z \notin E_{i+1} \mid z \in E_i] \cdot \mathbf{Pr}\left[\mathsf{Cut}_z^i \mid \{z \in E_i\} \wedge \{z \notin E_{i+1}\}\right]
\end{aligned}$$

Therefore, we conclude:

$$\mathbf{Pr}[\mathsf{Cut} \mid Y_i] \leq \frac{\mathbf{Pr}\left[\mathsf{Cut}_z^i \mid z \in E_i\right]}{\mathbf{Pr}[z \notin E_{i+1} \mid z \in E_i]} \leq O\left(\frac{\alpha \cdot \ell_z}{\beta \cdot \mathcal{D}}\right) \tag{10.36}$$

Plugging together our two bounds, the law of total probability now gives us:

$$\begin{aligned}
\mathbf{Pr}[\mathsf{Cut}] &\leq \lim_{T\to\infty} \sum_{i=1}^{T} \mathbf{Pr}[Y_i] \cdot \mathbf{Pr}[\mathsf{Cut} \mid Y_i] && \triangleright \textit{Law of Total Prob.} \\
&\leq \lim_{T\to\infty} \sum_{i=1}^{T} \mathbf{Pr}[Y_i] \cdot O\left(\frac{\alpha \cdot \ell_z}{\beta \cdot \mathcal{D}}\right) && \triangleright \textit{By Ineq. } (10.36) \\
&\leq O\left(\frac{\alpha \cdot \ell_z}{\beta \cdot \mathcal{D}}\right) \cdot \lim_{T\to\infty} \sum_{i=1}^{T} \mathbf{Pr}[Y_i] \\
&\leq O\left(\frac{\alpha \cdot \ell_z}{\beta \cdot \mathcal{D}}\right) \cdot \lim_{T\to\infty} \sum_{i=1}^{T} (1-\beta)^i && \triangleright \textit{By Ineq. } (10.35) \\
&\leq O\left(\frac{\alpha \cdot \ell_z}{\beta \cdot \mathcal{D}}\right) \cdot \frac{1}{\beta} \leq O\left(\frac{\alpha \cdot \ell_z}{\beta^2 \cdot \mathcal{D}}\right)
\end{aligned}$$

In the last line, we used that for each x with $\frac{|x-1|}{|x|} \leq 1$, it holds:

$$\lim_{T\to\infty} \sum_{i=1}^{T} (1 - {}^1\!/x)^{i-1} = \lim_{T\to\infty} \sum_{i=0}^{T} (1 - {}^1\!/x)^{i} = x$$

This is clearly the case for ${}^1\!/x := \beta \leq 1$. Thus, the proclaimed cutting probability of $O\left(\frac{\alpha \cdot \ell_z}{\beta^2 \cdot \mathcal{D}}\right)$ follows, which concludes the proof. □

The algorithm can be implemented in $\tilde{O}(1)$ minor aggregations and $\tilde{O}(1)$ $(1+\epsilon)$-approximate SetSSP computations with $\epsilon \in O\left({}^1\!/\log^2 n\right)$.

10.5 Low-Diameter Decompositions for k-path Separable Graphs

In this section, we want to prove the second statement from Theorem 8. More precisely, we want to show that we can create an LDD with quality $O(\log \log n)$ for any $\tilde{O}(1)$-path separable graph within $\tilde{O}(1)$ minor aggregations and $(1+\epsilon)$-approximate SetSSP's. To achieve this, we present an efficient distributed algorithm that creates an for k-path separable graphs with quality $O(\log k + \log \log n)$. Thus, it has a logarithmic dependence on k and $\log n$. The main technical result of this section is the following proposition that (one the first glance) is slightly weaker than what we want. We show that:

Lemma 10.14 (A Clustering Theorem for Restricted Graphs). *Let $\mathcal{D} > 0$ be an arbitrary distance parameter and $G := (V, E, \ell)$ be a (possibly weighted) $\tilde{O}(1)$-path separable graph. Then, there is an algorithm that creates a series of disjoint clusters $\mathcal{K} := K_1, K_2, \ldots, K_N$ with $K_i := (V_i, E_i)$. Futher, it holds:*

1. ***Strong Diameter:*** *Each cluster $K_i \in \mathcal{K}$ has a strong diameter of at most $\mathcal{D}$.*
2. ***Low Edge-Cutting Probability:*** *Each edge $z := \{v, w\} \in E$ of length ℓ_e is cut between clusters with probability (at most) $O\left(\frac{\ell_z \cdot (\log k + \log \log n)}{\mathcal{D}}\right)$.*

3. ***High Clustering Probability:*** *Each node $v \in V$ is added to some cluster $K_i \in \mathcal{K}$ with probability at least $1/2$.*

The algorithm can be implemented in $\tilde{O}(k)$ minor aggregations and $\tilde{O}(k)$ $(1+\epsilon)$-approximate SetSSP *computations with $\epsilon \in O\left(1/\log^2 n\right)$.*

Following this proposition, we can create a clustering where not all nodes are part of a cluster. In other words, we do not create a partition of the input graph G but leave some nodes unclustered. However, the diameter and edge-cutting probability are (asymptotically) correct, and we can guarantee that at least half of the nodes are clustered on expectation. Although this is not quite what we want, the proposition is sufficient to prove Theorem 8 for $\tilde{O}(1)$-separable graphs.

For $k \in \tilde{O}(1)$ one application of the algorithm creates a clustering with strong diameter $\mathcal{D}$ and cutting probability $O\left(\frac{\ell_z \cdot (\log\log n)}{\mathcal{D}}\right)$.

Recall that universally k-path separable graphs are closed under minor-taking. Therefore, the graph $G \setminus \mathcal{K}$ is also k-path separable, and we can apply the algorithm to $G \setminus \mathcal{K}$ with the same guarantees to cluster at least half of the remaining nodes. By Lemma 10.13, after $O(\log n)$ recursive applications, all nodes are clustered, and we obtain an LDD with quality $O(\log\log n)$ as required. For $k \in \tilde{O}(1)$ one iteration of the algorithm can, w.h.p., be implemented in $\tilde{O}(1)$ minor aggregations and $\tilde{O}(1)$ $(1+\epsilon)$-approximate SetSSP computations with $\epsilon \in O\left(1/\log^2 n\right)$. Thus, $O(\log n)$ recursive applications are within the required time complexity of Theorem 8.

Suppose we want to construct an LDD $\mathcal{K} = K_1, \dots, K_N$ with strong diameter $\mathcal{D}$ for some graph $G = (V, E)$. From a high-level perspective, the algorithm's execution proceeds in two distinct phases: the *backbone phase* and the *refinement phase*. In the first phase, we create a special partition $\mathcal{K}[\mathcal{B}] := K(\mathcal{B}_1), \dots, K(\mathcal{B}_N)$ of G. Each node $v \in K(\mathcal{B}_i)$ is in distance at most $\mathcal{D}_{BC} < \mathcal{D}$ to some subset $\mathcal{B}_i \subset K_i$ that consist of $\tilde{O}(k)$ (possibly disjoint) paths of bounded length of $O\left(\mathcal{D}\log^2\right)$. These paths may not necessarily be short**est** paths in G or even $K(\mathcal{B}_i)$, only their length is bounded. We call these subsets $\mathcal{B}_1, \dots, \mathcal{B}_N$ the *backbones* of the clusters. In contrast to an LDD, the clusters do **not** have simple nodes $v \in V$ as centers. As a result, the diameter of the resulting clusters might be much larger than $\mathcal{D}$. Moreover, a single cluster might not even be connected. The partition $\mathcal{K}[\mathcal{B}]$ can be constructed by a recursive divide-and-conquer algorithm executed in parallel on all connected components of G. The algorithm works as follows: Let $G_t \subset G$ be the graph at the beginning of the t^{th} recursive step where initially $G_0 = G$. At the beginning of each recursive step, we create an LDD $C_1, \dots, C_{N'}$ of diameter $(\mathcal{D}' \in O(\mathcal{D}\log^2 n)$ of G_t. We use the algorithm from Theorem 9 for this. All edges between two clusters C_i and C_j will not be considered in the following iterations. Then, in each C_i, we compute a weak $\tilde{O}(k)$-path $(\mathcal{D}', \epsilon)$-separator $\mathcal{S}_{\rangle}$. Recall that we defined this separator to be the union of $\tilde{O}(k)$ paths of length $2 \cdot \mathcal{D}'$ and some nodes in distance $\epsilon\mathcal{D}'$ to these paths. The $\tilde{O}(k)$ paths in $\mathcal{S}_i$ will be the backbone of a cluster. To add further nodes to the cluster, we first add all nodes at a random distance to $\mathcal{S}_i$ and then apply the blurry ball growing procedure from Lemma 10.7. This results in a cluster $K(\mathcal{B}_i) \subseteq C_i$ for each subgraph C_i. We describe the exact choice of parameters for these procedures in the detailed description of the analysis. Finally, we add all subgraphs from all clusters to our clustering $\mathcal{K}[\mathcal{B}]$ and remove them from the graph. We repeat this process with the remaing graph G_{t+1} until all subgraphs are empty. As we remove a $\mathcal{D}'$-separator in each step and create clusters of diameter $\mathcal{D}'$, the size of the largest connected components shrinks by a constant factor in each recursion. Thus, we require $O(\log n)$ iterations overall. As we will see in the analysis, the parameters to the subroutines can be chosen such that an edge is cut with probability at most $O(\frac{\log k \log n \cdot \ell_z}{\mathcal{D}_{BC}})$ and each node $v \in K(\mathcal{B}_i)$ is in the distance $\mathcal{D}_{BC}$ to a backbone $\mathcal{B}_i$.

In the second phase, we turn the resulting clustering into a LDD with connected clusters of strong diameter $\mathcal{D}$. We do so by computing an LDD $\mathcal{K}_i = K_{i_1}, \dots, K_{i_N}$ of strong diameter $\mathcal{D}$ in each cluster $K(\mathcal{B}_i)$. To construct $\mathcal{K}_i$, we again use the generic LDD construction algorithm from Theorem 10.3. We use a small subset of the nodes from each backbone $\mathcal{B}_i$ as possible cluster centers and exploit that only a few centers can cluster a given node. The algorithm works as follows: On each path $P \in \mathcal{B}_i$, mark a subset of nodes in the distance $\mathcal{D}_{BC}$ to each other and use them as potential cluster centers. As we limit the number of paths in $\mathcal{B}_i$ to $\tilde{O}(k)$ and the length of each path to $O(\mathcal{D}\log^2 n)$, the number of centers is limited to $\tilde{O}(k)$. Further, each node $v \in K_i$ has at least one center in distance $2\mathcal{D}_{BC}$ by consruction. We then use these marked nodes as centers $\mathcal{X}_i$ to the algorithm of Theorem 9. This creates the desired LDD $\mathcal{K}_i$ with strong diameter $16\mathcal{D}_{BC}$ of quality $O(\log k + \log\log n)$ in each cluster. The union of all LDD's $\mathcal{K}_1, \dots, \mathcal{K}_N$ is the desired LDD. By the union bound, an edge is cut with probability $O(\frac{\ell \cdot \log k \log n}{\mathcal{D}})$ in either of the two phases, which proves the algorithm when choosing $\mathcal{D}_{BC} = \mathcal{D}/16$.

Our algorithm is inspired by the works of Abraham, Gavoille, Gupta, Neiman, and Talwar [AGG+14, AGG+19a] and the subsequent improvement of their algorithm Filtser[Fil19]. Surprisingly, we do not require any fundamental changes to obtain an efficient distributed algorithm. Both algorithms essentially follow the same two-phase structure sketched above. They first constructed a partition around paths that are sampled in a certain way, and then they refined each partition individually. For reference, Abraham, Gavoille, Gupta, Neiman, and Talwar dubbed these subsets around which the partitions are constructed *skeletons* while Filtser used the term *r-core*. However, their key properties are (almost) the same. We use the different term *backbones* because we construct them slightly differently. Our contribution when comparing our algorithm to [AGG+14, AGG+19a, Fil19] is that we show a) that the exact shortest path can be replaced by approximate paths and the blurry ball growing procedure from Lemma 10.7 and, therefore, can be implemented efficiently and b) that we can use weak separators to parallelize the algorithm and archive a logarithmic recursion depth efficiently.

Now, we want to move from this high-level idea to the actual algorithm. First, we define the clustering constructed in the first phase as follows:

Definition 10.4 (Backbone Clustering). *Let $\mathcal{D} > 0$ be a distance parameter and $G := (V, E, \ell)$ a (possibly weighted) graph. Then, a (α, β, κ)-backbone clustering is a series of disjoint subgraphs $K(\mathcal{B}_1), \dots, K(\mathcal{B}_N)$ with $K(\mathcal{B}_i) = (V_i, E_i)$ where $V_1 \sqcup \ldots \sqcup V_N = V$. Further, the following three conditions hold:*

1. ***Strong Pseudo-Diameter:** Each node $v \in K(\mathcal{B}_i)$ is in distance at most $\mathcal{D}$ to $\mathcal{B}_i$.*
2. ***Low Edge-Cutting Probability:** An edge of length ℓ is cut between clusters with probability $O\left(\frac{\ell \cdot \alpha}{\mathcal{D}}\right)$.*
3. ***Backbone Property:** Each backbone $\mathcal{B}_i$ consists of at most $O(\kappa)$ paths of length $O(\beta\mathcal{D})$.*

Following this definition, in the backbone phase, we create an (α, β, κ)-backbone clustering with $\alpha \in O(\log(k \log n))$, $\beta \in O(\log^2)$, $\kappa \in O(k \log^2 n)$ and pseudo-diameter $\mathcal{D}_{BC} \in O(\mathcal{D})$ for G. The core idea is to recursively create weak $O(k \log n)$-path $(D \log^2 n, O(\frac{1}{\log^2 n})$-separators and use BBG to carve clusters around the separators. To ensure short paths and a low recursion depth, we apply the generic LDD with parameter $O(\mathcal{D} \log^2 n)$ after carving each cluster.

We formalize this in the following lemma:

Lemma 10.15 (Backbone Clustering). *Let $\mathcal{D} > 0$ be a distance parameter and $G := (V, E, \ell)$ a (possibly weighted) k-path separable graph. Then, there is an algorithm that creates an (α, β, κ)-backbone clustering with $\alpha \in O(\log(k \log n))$, $\beta \in O(\log^2)$, $\kappa \in O(k \log^2 n)$ and pseudo-diameter $\mathcal{D}$ for G.*

The algorithm can be implemented with $\tilde{O}(1)$ minor aggregations and $(1+\epsilon)$-approximate SetSSP computations where $\epsilon \in O\left(1/\log^2 n\right)$.

That means we partition graph G into subgraphs $K_1, \dots, K_N$ such that each K_i contains a backbone $\mathcal{B}_i$ of $O(\log n)$ paths with length at most $O(\mathcal{D} \log n)$. Further, each node $v \in K_i$ is in the distance $O(\mathcal{D})$ to at least one of the paths in $\mathcal{B}_i$. Finally, each edge is cut with probability $O(\frac{\ell \cdot \log \log n}{\mathcal{D}})$. Thus, the resulting clustering has the correct cutting probability for the edges, but the diameter might be too high. This follows because we cannot argue that two nodes that are in distance $O(\mathcal{D})$ to some set $\mathcal{B}_i$ are in distance at most $O(2\mathcal{D})$ to one another.

In the second phase, we must, therefore, reduce the clusters' diameters. To this end, we exploit the specific structure of the backbone clusters to derive clusters of strong diameter $\mathcal{D}$; More precisely, for the second phase, we show a generic lemma that allows us to turn any backbone clustering into an LDD. To be precise, it holds:

Lemma 10.16. *Suppose we have an algorithm $\mathcal{A}$ that creates a (α, β, κ)-backbone clustering with pseudo-diameter $\mathcal{D}_{BC}$ of a weighted graph G. Then, there is an algorithm that creates a series of disjoint clusters $\mathcal{K} := K_1, \dots, K_N$ with $K_i := (V_i, E_i)$. Futher, it holds:*

1. ***Strong Diameter:** Each cluster $K_i \in \mathcal{K}$ has a strong diameter of at most $16 \cdot \mathcal{D}_{BC}$.*
2. ***Low Edge-Cutting Probability:** Each edge $z := \{v, w\} \in E$ of length ℓ_e is cut between clusters with probability (at most) $O\left(\frac{\ell_z \cdot (\alpha + \log \kappa\beta)}{\mathcal{D}}\right)$.*
3. ***High Clustering Probability:** Each node $v \in V$ is added to some cluster $K_i \in \mathcal{K}$ with probability at least $1/2$.*

*If each path in the backbone is an **exact** shortest path in its cluster, the length of the paths does affect the quality and the cutting probability improves to $O\left(\frac{\ell_z \cdot (\alpha + \log \kappa\beta)}{\mathcal{D}}\right)$.*

The algorithm requires one application of $\mathcal{A}$, $\tilde{O}(\kappa)$ minor aggregations, and $\tilde{O(1)}$ approximate SetSSP computations with $\epsilon \in O(1/\log^2 n)$.

In our algorithm, we use the algoriithm promised by this lemma in the following way: We construct a backbone clustering that consists of $O(k \log^2 n)$ paths of length $O(\mathcal{D} \log^2)$ and each node is in distance $(4/100) \cdot \mathcal{D}$ to these paths. With Lemma 10.16 from the previous section, this backbone clustering can be turned into an LDD. The diameter of the resulting clusters is $12 \cdot (4/100) \cdot \mathcal{D} \leq \mathcal{D}$. Note that for our clustering, it holds $\alpha \in O(\log(k \log n))$ and $\kappa\beta \in O(k \log^4)$. Therefore, the LDD created by Lemma 10.16 has quality

$O(\log(k \log n))$. As both the algorithms from Lemma 10.16 and Lemma 10.15 rely on $\tilde{O}(1)$ aggregation and SetSSP computations, the theorem follows.

Following the algorithm's two-phase structure, we divide the remainder of this section into two parts, focusing on the backbone and the refinement phase. The main challenge is finding an appropriate clustering algorithm for the first phase, as its carefully chosen properties more or less directly imply the correctness of the second. We will first describe and analyze the algorithm behind Lemma 10.15 in Subsection 10.5.1. Then, we describe the refinement in Section 10.5.2 and prove Lemma 10.16.

10.5.1 The Backbone Clustering Phase (Proof of Lemma 10.15)

In this section, we prove the following technical lemma which that that can can efficiently constuct backbone clusters using approximate shortest paths and minor aggregations.

Lemma 10.15 (Backbone Clustering). *Let $\mathcal{D} > 0$ be a distance parameter and $G := (V, E, \ell)$ a (possibly weighted) k-path separable graph. Then, there is an algorithm that creates an (α, β, κ)-backbone clustering with $\alpha \in O(\log(k \log n))$, $\beta \in O(\log^2)$, $\kappa \in O(k \log^2 n)$ and pseudo-diameter $\mathcal{D}$ for G.*

The algorithm can be implemented with $\tilde{O}(1)$ minor aggregations and $(1 + \epsilon)$-approximate SetSSP computations where $\epsilon \in O\left({}^{1}/_{\log^2 n}\right)$.

The algorithm behind this lemma is a divide-and-conquer algorithm executed in parallel on all connected components of G. For the *divide* step, we use the LDD promised by Theorem 10 to create disjoint subgraphs of diameter $\mathcal{D}' \in O(\mathcal{D} \log^2 n)$. The *conquer* step works in five synchronized phases that compute and remove a weak $(\mathcal{D}', \epsilon)$-separator from each of these subgraphs. Further, we create a backbone cluster from each separator. This works in two steps, by first adding all nodes at a random distance and then applying the blurry ball growing procedure from Lemma 10.7. We repeat this process until all subgraphs are empty. As we remove a $\mathcal{D}'$-separator in each step and create clusters of diameter $\mathcal{D}'$, this requires $O(\log n)$ iterations.

After this intuition, we move to the detailed description of the algorithm. For this, we define some useful notations/constants, namely

$$\begin{aligned}
\mathcal{D}' &= 100 \cdot \mathcal{D} \cdot \log^2 n \\
\epsilon &= \frac{1}{10000 \log^2 n} \\
\mathcal{D}_{BC} &= \frac{\mathcal{D}}{4} \\
\rho_{\mathsf{blur}} &:= \frac{\mathcal{D}}{c_{\mathsf{blur}} \cdot \log\log n} \leq \frac{\mathcal{D}}{8}
\end{aligned}$$

Here, $c_{\mathsf{blur}} > 8$ is a large constant that will be determined in the analysis. In the following, we will consider a single recursive step $t \in [1, O(\log n)]$ of the algorithm. We will slightly abuse notation and denote the set of all nodes that have already been added to some cluster as $\mathcal{K}_t$ Further, we call a node $v \in V \setminus \mathcal{K}_t$, which is not yet part of a cluster, an *uncharted* node. A single recursive *conquer* step works on the graph $G_t = G \setminus \mathcal{K}_t$ of uncharted nodes works as follows:

(Step 1) Create Partitions: Let $C_1, \dots, C_N$ be the connected components of G_t. Compute an LDD with diameter $\mathcal{D}'$ in each subgraph C_i using the algorithm from Theorem 10. The resulting partitions are connected subgraphs $P_1, \dots, P_{N'}$ with diameter $\mathcal{D}'$.

(Step 2) Create Weak Separators in All Partitions: In each partition P_i, we compute a weak $(\mathcal{D}', \epsilon)$-separator S_i using the algorithm from Theorem 6. As the distance parameter for the separator, we choose $\mathcal{D}'$, and for the approximation parameter, we pick ϵ.

(Step 3) Create Random Ball Around Separator: We carve a ball with random diameter $O(X_i \cdot \mathcal{D}_{BC})$ where $X_i \sim \mathsf{Texp}(4 \cdot \log\log n)$. To this end, we perform a $(1+\epsilon)$-approximate SetSSP for S_i. Thereby, we obtain an a $(1+\epsilon)$-approximate SetSSP tree T_i and all nodes $w \in P_i$ know a value $d_{T_i}(w, S_i)$, which is its $(1+\epsilon)$-approximate distance to S_i in P_i. After that, we mark every node in the set:

$$K_{\mathsf{Texp}}(S_i) := \{w \in P_i \mid d_{T_i}(w, S_i) \leq (1+\epsilon) \cdot X_i \cdot \mathcal{D}_{BC}\}$$

(Step 4) Blur the Ball: Finally, we apply the BBG procedure of Lemma 10.7 to the ball $K_{\mathsf{Texp}}(S_i)$ in each partition. As the distance parameter for this procedure, we choose ρ_{blur}. Thus, for each set $K_{\mathsf{Texp}}(S_i)$ we obtain the superset $K_{\mathsf{blur}}(S_i) := \mathsf{blur}\left(K_{\mathsf{Texp}}(S_i), \rho_{\mathsf{blur}}\right)$. Here, $\mathsf{blur}(S, \rho)$ is an application of BBG from Lemma of 10.7 to a set S with parameter ρ.

(Step 5) Prepare Next Recursion: We choose each $K_i = K_{\mathsf{blur}}(S_i)$ as a backbone cluster and add K_i to $\mathcal{K}$. Each node in K_i marks itself as inactive and removes itself from future iterations. All edges between active nodes and nodes in K_i are marked as *cut*.

We will now present the proof of Lemma 10.15. Recall that in this phase we want to efficintly construct an $\left(O(\log(k \log n)), O(\log^2), O(k \log^2 n)\right)$-backbone clustering with pseudo-diameter $\mathcal{D}_{BC} = (4/100) \cdot \mathcal{D}$ for G. The proof is divided into four parts. We begin by showing that the algorithm produces a clustering that fulfills all conditions of a backbone clustering, namley the backbone property (Lemma 10.17), the pseudo-diameter (Lemma 10.18), and the cutting probability (Lemma 10.19). Finally, we prove the time complexity in Lemma 10.24. Thus, altogther Lemma 10.24 and Lemmas 10.17, 10.18, and 10.19 imply Lemma 10.15.

Backbone Property First, we note that, indeed, each cluster has a backbone that consists of a few short paths. However, it follows directly from the construction. For the sake of completeness, we note that it holds:

Lemma 10.17 (Backbone Property). *Each cluster contains a backbone $\mathcal{B}_i$ consists of at most $O(k \log^3 n)$ paths of length $O(\mathcal{D} \log^2 n)$, w.h.p.*

This follows directly from the guarantees of our weak separator. Note that the LDD between each step ensures that each subgraph has a diameter $\mathcal{D}'$. Recall that by Theorem, we can construct a weak $O(\epsilon^{-1} \cdot k \cdot$

$\log n)$-path $(\mathcal{D}, \epsilon)$-separator, for a weighted k-path separable graph $G := (V, E, \ell)$ with weighted diameter smaller than $\mathcal{D}'$. For $\epsilon \in \Theta(1/\log^2 n)$, this separator has $O(k \log^3 n)$ paths of length $O(\mathcal{D} \log^2 n)$, w.h.p. We choose these paths as the backbone, which proves the lemma.

Pseudo Diameter Second, we observe the *diameter* of each cluster, i.e., the distance of each node to the closest backbone is $\mathcal{D}$. It holds:

Lemma 10.18 (Pseudo-Diameter). *Each node $v \in K(\mathcal{B}_i)$ is in distance at most $\mathcal{D}$ to $\mathcal{B}_i$.*

Proof. For the second statement, consider the $O(k \log^3 n)$ paths in the separator and denote this set as $\mathcal{B}_i$. Each node in separator S_i is in the distance at most $\epsilon\mathcal{D}'$ to one of these paths. This follows from the definition of a weak $(\mathcal{D}', \epsilon)$-separator. Our choice of ϵ and $\mathcal{D}'$ yields:

$$\epsilon\mathcal{D}' \leq \frac{100\mathcal{D}\log^2 n}{10000 \cdot \log^2 n} = \frac{\mathcal{D}}{100} \leq \frac{\mathcal{D}}{4} = \mathcal{D}_{BC} \tag{10.37}$$

Further, each node in $K_{\mathsf{Texp}}(S_i)$ is in the distance at most $(1+\epsilon) \cdot X_t \cdot \mathcal{D}_{BC} \leq 2\mathcal{D}_{BC}$ as to S_i. This follows because $X_t \in [0, 1]$ and $\epsilon \leq 1$. Finally, each node $v \in K_{\mathsf{blur}}(S_i)$ is in the distance at most $\mathcal{D}_{BC}$ to $K_{\mathsf{Texp}}(S_i)$ by our choice of ρ and guarantees of Lemma 10.7. Following Lemma 10.7, the process adds nodes in distance $\frac{\rho}{1-\alpha}$ where $\alpha \in O(\log\log n/\log n)$. For a large enough n and $c_{\mathsf{blur}} \geq 4$, we have

$$\frac{\rho}{1-\alpha} \leq \frac{\mathcal{D}}{c_{\mathsf{blur}} \cdot \log\log n(1 - O(\log\log n/\log n))} \leq \frac{\mathcal{D}}{4} = \mathcal{D}_{BC} \tag{10.38}$$

Summing up, let $\mathcal{B}_i$ the set of paths in S_i, then it holds:

$$\begin{aligned}
d(u, \mathcal{B}_i) &\leq d(u, K_{\mathsf{Texp}}(S_i)) + \max_{v \in K_{\mathsf{Texp}}(S_i)} d(v, \mathcal{B}_i) \\
&\leq d(v, K_{\mathsf{Texp}}(S_i)) + \max_{v \in K_{\mathsf{Texp}}(S_i)} d(v, S_i) + \max_{w \in S_i} d(w, \mathcal{B}_i) \\
&\leq \mathcal{D}_{BC} + 2\mathcal{D}_{BC} + \mathcal{D}_{BC} \\
&= \frac{4\mathcal{D}}{4} = \mathcal{D}
\end{aligned}$$

Therefore, the total distance from a node in $K_{\mathsf{blur}}(S_i)$ to a path in S_i is at most $\mathcal{D}$. □

Cut Probability Next, we show the cut probability, which perhaps has the most challenging proof of this chapter. We prove that the following holds:

Lemma 10.19. *An edge $z \in E$ is cut with probability at most $O\left(\frac{\alpha\ell_z}{\mathcal{D}}\right)$.*

For each edge $z \in E$, two operations can cut an edge in each iteration. The decomposition from Theorem 10 and the ball $K_{\mathsf{blur}}(S_i)$, which is created through *blurry ball growing*. The other possibility is that z is cut by $K_{\mathsf{blur}}(S_i)$. Again, there are at most $O(\log n)$ sets S_i whose ball $K_{\mathsf{blur}}(S_i)$ can potentially cut it. By our choice of ρ, the probability for the ball cut to z is $O\left(\ell_z \log\log n/\mathcal{D}\right)$. Therefore, we cannot simply use the union bound to sum up the probabilities over all iterations. However, we can exploit that S_i must be *close*, i.e., within distance $O(\rho)$, to either endpoint of z if $K_{\mathsf{blur}}(S_i)$ can possibly cut it. In such a case, both endpoints z are already added $K_{\mathsf{Texp}}(S_i)$ with constant probability, i.e., they are safe before the blur procedure is even executed. This follows from the properties of the truncated exponential distribution. Thus, on expectation, there will only be a constant number of tries before z is either cut or safe. This is sufficient for our probability bound.

Having established the rough idea, let us now prove the lemma in detail. We define the so-called $(\lambda, c_{\mathsf{blur}}, c_{\mathsf{ldd}})$-bounded Blurry Ball Carving Processes that resembles our algorithm:

Definition 10.5 ($(\lambda, c_{\mathsf{blur}}, c_{\mathsf{ldd}})$-bounded Blurry Ball Carving Process). *Let $G := (V, E, w)$ a weighted graph and let $\mathcal{D}_{BC}$ be a distance parameter. Then, a $(\lambda, c_{\mathsf{blur}}, c_{\mathsf{ldd}})$-bounded blurry ball carving process with parameters $\lambda \geq 1$ and $c_{\mathsf{blur}}, c_{\mathsf{ldd}} \geq 0$ creates a series of subgraphs $G := G_0, G_1, \ldots$, vertex sets $A_1, A_2, \ldots$, and clusters $K_1, K_2, \ldots$, s.t., it holds:*

1. *$K_t \supseteq A_t$ is a superset of the ball $B_{G_{t-1}}(A_t, X_t \cdot \mathcal{D}_{BC})$ with $X_t \sim \mathsf{Texp}(\lambda)$, i.e., it holds:*

$$K_t \supseteq B_{G_{t-1}}(A_t, X_t \cdot \mathcal{D}_{BC}) \qquad \textbf{(Property 1.a)}$$

Further, it holds that:

$$K_t \subseteq B_{G_{t-1}}\left(A_t, \left(X_t + \frac{1}{2 \cdot c_{\mathsf{blur}} \cdot \lambda}\right) \cdot \mathcal{D}_{BC}\right) \qquad \textbf{(Property 1.b)}$$

For all edges $(v, w) := z \in E_t$ of length ℓ, it holds:

$$\mathbf{Pr}[v \in K_t, w \notin K_t] \in O\left(\frac{c_{\mathsf{blur}} \cdot \lambda \cdot \ell}{\mathcal{D}_{BC}}\right) \qquad \textbf{(Property 1.c)}$$

2. *$G_t \subseteq G_{t-1} \setminus K_t$ is a (random) decomposition of $G_{t-1} \setminus K_t$ that only removes edges. Further, for each step t there is value $\alpha(G_{t-1})$ that depends only on G_{t-1} and t, s.t., it holds:*

$$\mathbf{Pr}[e \notin E(G_t) \mid e \in E(G_{t-1} \setminus K_t)] \leq \alpha(G_{t-1}) \cdot \frac{\ell_e}{\mathcal{D}_{BC}} \qquad \textbf{(Property 2.a)}$$

Finally, let Ω_t be the set of all sequences of graphs that can be created by the process until step t, then

$$\max_{T<n} \max_{G_1,\ldots,G_T \in \Omega_T} \sum_{t=1}^{T} \alpha(G_t) \leq c_{\mathsf{ldd}} \cdot \lambda \qquad \textbf{(Property 2.b)}$$

*If we choose K_t based on the (approximate) distances in G and not in G_{t-1} we call it a **weak** ball carving process. Note that the set $A_t \subseteq V_{t-1}$ can be freely chosen from the unclustered nodes.*

Lemma 10.20. *The algorithm from Section 10.5.1 is a $(\lambda, c_{\mathsf{blur}}, c_{\mathsf{ldd}})$-bounded blurry ball carving process with $\lambda = 4 \log\log n$, $c_{\mathsf{blur}} = 250$, and $c_{\mathsf{ldd}} \in O(1)$.*

Proof. We prove that it fulfills all 5 properties. The sets $A_1, A_2, \ldots$ are the separators that we sample in each recursive step.

- **Property 1.a**: This property is fulfilled because K_{Texp} adds all nodes in approximate distance $X_i \cdot \frac{\mathcal{D}}{100}$. Let w a node in distance at most $X_i \cdot \frac{\mathcal{D}}{100}$ to S_i. By the guarantees of our approximation algorithm, it holds:

$$d_{P_i}(w, A_t) \leq (1+\epsilon) \cdot d_{T_i}(w, S_i) \leq (1+\epsilon) \cdot X_i \cdot \mathcal{D}_{BC}$$

- **Property 1.b**: This property is fulfilled due to the use of blurry ball growing with parameter $\rho \in O\left(\frac{c_{\mathsf{blur}}\lambda}{\mathcal{D}}\right)$. It holds:

$$d(w', A_t) \leq d(w', K_{\mathsf{Texp}}(A_t)) + \max_{w \in K_{\mathsf{Texp}}(A_t)} d(w, A_t)$$

As we overestimate:

$$\leq d(w', K_{\mathsf{Texp}}(A_t)) + (1+\epsilon) \cdot X_i \cdot \mathcal{D}_{BC}$$

By Lemma 10.7:

$$\leq \left(1 - O\left(\frac{\log\log n}{\log n}\right)^{-1}\right) \cdot \frac{\mathcal{D}_{BC}}{250 c_{\mathsf{blur}}\lambda}$$
$$+ (1+\epsilon) \cdot X_i \cdot \mathcal{D}_{BC}$$

For $O\left(\frac{\log\log n}{\log n}\right) \leq \frac{1}{2}$:

$$\leq \frac{\mathcal{D}_{BC}}{125 c_{\mathsf{blur}}\lambda} + (1+\epsilon) \cdot X_i \cdot \mathcal{D}_{BC}$$

As $X_t \leq 1$:

$$\leq \frac{\mathcal{D}_{BC}}{125 c_{\mathsf{blur}}\lambda} + (X_t + \epsilon) \cdot \mathcal{D}_{BC}$$
$$\leq \left(X_i + \frac{1}{125 c_{\mathsf{blur}}\lambda} + \epsilon\right) \cdot \mathcal{D}_{BC}$$

For $\epsilon \leq \frac{\mathcal{D}_{BC}}{125 c_{\mathsf{blur}}\lambda}$:

$$\leq \left(X_i + \frac{1}{60 c_{\mathsf{blur}}\lambda}\right) \cdot \mathcal{D}_{BC}$$

- **Property 1.c**: This property is also fulfilled due to the use of blurry ball growing with parameter $\rho \in O\left(\frac{c_{\mathsf{blur}}\lambda}{\mathcal{D}}\right)$ (cf. Lemma 10.7).
- **Property 2.a**: As we pick $\mathcal{D}' \in O(\mathcal{D}\log^2 n)$, a single application of Theorem 10 cuts an edge with probability $O\left(\ell_z \log n / \mathcal{D}'\right) = O\left(\ell_z / \mathcal{D} \log n\right)$. Therefore, we gave $\alpha(G_{t-1}) \in O(1/\log n)$ in each recursive step
- **Property 2.b**: Recall that we perform at most $O(\log n)$ recursive step until all nodes are clustered. Further, we have $\alpha(G_{t-1}) \in O(1/\log n)$ for each graph, no matter the topology or size. Thus, as we apply the decomposition $O(\log n)$ times, the parameters sum up to $O(1)$.

This proves the lemma. □

Now fix an edge $z \in E$. During this proof, we will only consider the connected component that contains z. Thus, when we talk about a step A_t or a cluster K_t, we mean the set or cluster in z's component. For convenience, we further define the following terms:

$$\lambda := 4 \log\log n \tag{10.39}$$

$$\gamma_z := \frac{\ell_z}{\mathcal{D}_{BC}} \tag{10.40}$$

$$\gamma'_{\mathsf{blur}} = \frac{1}{2c_{\mathsf{blur}}\lambda} \tag{10.41}$$

First, we note that only because a cluster around a center A_t could **potentially** cut z, it does not automatically mean that it will happen. If an edge is not cut, there are two possibilities. Either both endpoints remain active (and will be clustered in a future iteration), or both endpoints of the edge could be added to the cluster K_t. In the latter case, the edge would be *safe* as it can never be cut in future steps. Our goal is now to bound the probability that an edge is safe. We begin this part of our analysis with a useful observation that quantifies under which circumstances a node is added to a cluster and when it is *safe* from being clustered. Recall that by (**Property 1.a**), the probability that both endpoints of z is added to K_t if

$$d_{G_t}(z, A_t) < X_t \cdot \mathcal{D}_{BC} - \gamma_z \cdot \mathcal{D}_{BC}. \tag{10.42}$$

Further, by (**Property 1.b**), for any node $v \in V$ added added to K_t, it must hold:

$$d_{G_t}(v, A_t) \leq X_t \cdot \mathcal{D}_{BC} + \gamma'_{\mathsf{blur}} \cdot \mathcal{D}_{BC} \tag{10.43}$$

Going on, we say that an edge is *threatened* by some set A_t if

$$\mathcal{C}_t := \underbrace{\{d_{G_t}(z, A_t) \geq X_t \cdot \mathcal{D}_{BC} - \gamma_z \cdot \mathcal{D}_{BC}\}}_{z\text{ is not fully added to the cluster.}} \cap \underbrace{\{d_{G_t}(z, A_t) \leq X_t \cdot \mathcal{D}_{BC} + \gamma'_{\mathsf{blur}} \cdot \mathcal{D}_{BC}\}}_{z\text{ may be added.}}$$

In this case, the edge is **potentially** affected by the blurry growing procedure and may be cut. Note that it holds for every edge $z \in E$ by **Property 1.c** and **Property 2.a** that:

$$\mathbf{Pr}\left[\mathsf{Cut}_z^t \mid \mathcal{C}_t\right] \leq \underbrace{O\left(\frac{c_{\mathsf{blur}} \cdot \lambda \cdot \ell_z}{\mathcal{D}_{BC}}\right)}_{z\text{ cut by }K_t.} + \underbrace{\alpha(G_{t-1}) \cdot \frac{\ell_z}{\mathcal{D}_{BC}}}_{z\text{ cut by decomposition.}} \tag{10.44}$$

Otherwise, it can only be cut by the decomposition, so by **Property 2.a** we only have

$$\mathbf{Pr}\left[\mathsf{Cut}_z^t \mid \overline{\mathcal{C}}_t\right] \leq \underbrace{\alpha(G_{t-1}) \cdot \frac{\ell_z}{\mathcal{D}_{BC}}}_{z\text{ cut by decomposition.}} \tag{10.45}$$

We define the set of *threateners* as:

$$\mathcal{I}_z := \{A_t \mid d_{G_t}(z, A_t) \leq (1+\gamma'_{\mathsf{blur}}) \cdot \mathcal{D}_{BC}\} \tag{10.46}$$

We will show that we can bound the probability of an edge being cut based on the expected number of threateners. Note that this is slightly more general than we need it to be. It holds:

Lemma 10.21. *Consider a $(\lambda, c_{\mathsf{blur}}, c_{\mathsf{ldd}})$-bounded blurry ball growing process and assume that for some universal constants c_1, c_2 that are independent of λ, it holds:*

$$\mathbb{E}\left[|\mathcal{I}_z|\right] \leq c_1 \cdot e^{(\lambda+c_2)\cdot\left(1+\gamma'_{\mathsf{blur}}\right)} \tag{10.47}$$

Then, for $\lambda_z \in O\left(\mathcal{D}_{BC}/\lambda\right)$, it holds:

$$\mathbf{Pr}[\mathsf{Cut}] \leq O\left(\frac{(c_{\mathsf{blur}} + c_{\mathsf{ldd}})\lambda \cdot \ell_z}{\mathcal{D}_{BC}}\right) \tag{10.48}$$

Note that in the algorithm of Section 10.5.1 at most $O(\log n)$ sets can threaten an edge z on expectation. This follows, as we have at most $O(\log n)$ iterations, w.h.p., and there is at most one threatener per iteration. Further, recall that we choose $\lambda = 4 \log\log n$. Therefore, we have:

$$c_1 \cdot e^{(\lambda+c_2)\cdot\left(1+\gamma'_{\mathsf{blur}}\right)} \geq e^{\lambda} = e^{4\log\log n} = \log^4(n) \geq O(\log n)$$

Thus, Lemma 10.21 proves Claim 10.19.

We will show that if an edge is endangered, the probability that both endpoints of the edge itself are added to the cluster is constant. Thus, with a constant probability, the edge is safe from being cut in any future iteration. Intuitively, this follows due to the exponential distribution. If the random diameter is big enough to cluster a node close to z, it will likely also cluster z. The proof is of course more difficult as we need to account for the fact that we truncate the distribution and the imprecision introduced by the approximate shortest paths. We show that probability for this event depends on the **expected** number of centers that get close to z and - rather surprisingly - not on the exact number. It holds:

Lemma 10.22. *The probability that an edge is endangered in any of the τ steps we consider is:*

$$\sum_{t=0}^{\tau} \mathbf{Pr}[\mathcal{C}_t] \leq \left(1 - e^{-\lambda(\gamma_z + \gamma'_{\mathsf{blur}})}\right) \cdot e^{\lambda\gamma'_{\mathsf{blur}}} \cdot \left(1 + \frac{\mathbb{E}\left[|\mathcal{I}_z|\right]}{e^{\lambda} - 1}\right) \tag{10.49}$$

Proof. Consider a step t. Let A_t be the center around which a ball is carved and let X_t be the exponentially distributed random distance picked by A_t. Further denote

$$\rho_t := \frac{d_{G_t}(A_t, z)}{\mathcal{D}_{BC}} \tag{10.50}$$

as the normalized distance between A_t and z in G_t. Note that G_t and therefore also this distance depends on the random decisions of all previous rounds, so ρ_t is a random variable. In the case that z is already clustered by step t, we define the distance to be infinite. We now show that:

Claim 17. *It holds:*

$$\begin{aligned}&\mathbf{Pr}[\rho_t - \gamma'_{\mathsf{blur}} \leq X_t \leq \rho_t + \gamma_z \mid \rho_t]\\ &\leq \left(1 - e^{-\lambda(\gamma'_{\mathsf{blur}}+\gamma_z)}\right) \cdot e^{\lambda\gamma'_{\mathsf{blur}}} \cdot \left(\mathbf{Pr}[X_t \geq \rho_t \mid \rho_t] + \frac{1}{e^{\lambda}-1}\right)\end{aligned}$$

Proof. The proof follows directly from the definition of X_t. First, we note that it holds that

$$\mathbf{Pr}[X_t > \rho_t - \gamma'_{\mathsf{blur}}] = \int_{\rho_t-\gamma'_{\mathsf{blur}}}^{1} \frac{\lambda \cdot e^{-\lambda y}}{1-e^{-\lambda}} dy = \frac{e^{-(\rho_t-\gamma'_{\mathsf{blur}})\cdot\lambda} - e^{-\lambda}}{1-e^{-\lambda}} \tag{10.51}$$

$$= \frac{e^{-(\rho_t-\gamma'_{\mathsf{blur}})\cdot\lambda}}{1-e^{-\lambda}} - \frac{e^{-\lambda}}{1-e^{-\lambda}} \tag{10.52}$$

$$= \frac{e^{-(\rho_t-\gamma'_{\mathsf{blur}})\cdot\lambda}}{1-e^{-\lambda}} - \frac{1}{e^{\lambda}-1} \tag{10.53}$$

On the other hand, it holds:

$$\begin{aligned}&\mathbf{Pr}[\rho_t - \gamma'_{\mathsf{blur}} \leq X_t \leq \rho_t + \gamma_z \mid \rho_t]\\ &= \int_{\rho_t-\gamma'_{\mathsf{blur}}}^{\min\{1,\rho_t+\gamma_z\}} \frac{\lambda \cdot e^{-\lambda y}}{1-e^{-\lambda}} dy\\ &\leq \frac{e^{-(\rho_t-\gamma'_{\mathsf{blur}})\cdot\lambda} - e^{-(\rho_t+\gamma_z)\cdot\lambda}}{1-e^{-\lambda}}\end{aligned}$$

Substitute $\rho' = \rho_t - \gamma'_{\mathsf{blur}}$

$$\begin{aligned}&\leq \frac{e^{-\rho'\cdot\lambda} - e^{-\left(\rho'+\gamma'_{\mathsf{blur}}+\gamma_z\right)\cdot\lambda}}{1-e^{-\lambda}}\\ &\leq \left(1 - e^{-\lambda(\gamma'_{\mathsf{blur}}+\gamma_z)}\right) \frac{e^{-\lambda\rho'}}{1-e^{-\lambda}}\end{aligned}$$

Substitute back

$$\begin{aligned}&\leq \left(1 - e^{-\lambda(\gamma'_{\mathsf{blur}}+\gamma_z)}\right) \frac{e^{-\lambda(\rho_t-\gamma'_{\mathsf{blur}})}}{1-e^{-\lambda}}\\ &\leq \left(1 - e^{-\lambda(\gamma'_{\mathsf{blur}}+\gamma_z)}\right) \frac{e^{-\lambda(\rho_t-\gamma'_{\mathsf{blur}})}}{1-e^{-\lambda}} - \frac{1}{e^{\lambda}-1} + \frac{1}{e^{\lambda}-1}\\ &\leq \left(1 - e^{-\lambda(\gamma'_{\mathsf{blur}}+\gamma_z)}\right) \cdot e^{\lambda\gamma'_{\mathsf{blur}}} \cdot \frac{e^{-\lambda\rho_t}}{1-e^{-\lambda}} - \frac{1}{e^{\lambda}-1} + \frac{1}{e^{\lambda}-1}\\ &\leq \left(1 - e^{-\lambda(\gamma'_{\mathsf{blur}}+\gamma_z)}\right) \cdot e^{\lambda\gamma'_{\mathsf{blur}}} \cdot \left(\mathbf{Pr}[X_t > \rho_t \mid \rho_t] + \frac{1}{e^{\lambda}-1}\right)\end{aligned}$$

Thus, the claim follows. □

Next, Let $\mathcal{F}_t$ be the event that K_t contains (at least) one endpoint of edge z. This is clearly the case if the random variable X_t is at least ρ_t and we have

$$\mathbf{Pr}[\mathcal{F}_t] \geq \mathbf{Pr}[X_t \geq \rho_t \mid \rho_t]$$

Further, we define the following helper variables:

$$\begin{aligned}
\alpha &= e^{-\lambda(\gamma'_{\mathsf{blur}}+\gamma_z)} \\
\beta &= e^{\lambda\gamma'_{\mathsf{blur}}} \\
\zeta &= (1-\alpha)\beta
\end{aligned}$$

Note that by Lemma 17, we have the following relationship between $\mathcal{F}_t$ and $\mathcal{C}_t$:

$$\begin{aligned}
\mathbf{Pr}[\mathcal{C}_t \mid \rho_t] &\leq \mathbf{Pr}[\rho_t - \gamma'_{\mathsf{blur}} \leq X_t \leq \rho_t + \gamma_z \mid \rho_t] && \textit{By Definition if } \mathcal{C}_t. \\
&\leq \left(1 - e^{-\lambda(\gamma'_{\mathsf{blur}}+\gamma_z)}\right) \cdot e^{\lambda\gamma'_{\mathsf{blur}}} \cdot \left(\mathbf{Pr}[X_t \geq \rho_t \mid \rho_t] + \frac{1}{e^{\lambda}-1}\right) && \textit{By Claim 17.} \\
&:= \zeta \cdot \left(\mathbf{Pr}[X_t \geq \rho_t \mid \rho_t] + \frac{1}{e^{\lambda}-1}\right) && \textit{By Definition of } \zeta.
\end{aligned}$$

Thus, by definition of $\mathcal{F}_t$, it holds:

$$\mathbf{Pr}[\mathcal{C}_t \mid \rho_t] \leq \zeta \cdot \left(\mathbf{Pr}[\mathcal{F}_t \mid \rho_t] + \frac{1}{e^{\lambda}-1}\right) \tag{10.54}$$

In the following, let $f(\cdot)$ be the probability density function of ρ_t. By the law of total probability, we have:

$$\begin{aligned}
\mathbf{Pr}[\mathcal{C}_t \mid A_t \in \mathcal{I}_z] &:= \mathbf{Pr}[\mathcal{C}_t \mid \rho_t \leq 1 + \gamma'_{\mathsf{blur}}] && \textit{By Definition of } A_t. \\
&:= \int_0^{1+\gamma'_{\mathsf{blur}}} \mathbf{Pr}[\mathcal{C}_t \mid \rho_t = x] f(x)dx && \textit{By Law of Tot. Prob.} \\
&\leq \zeta \cdot \int_0^{1+\gamma'_{\mathsf{blur}}} \left(\mathbf{Pr}[\mathcal{F}_t \mid \rho_t = x] + \frac{1}{e^{\lambda}-1}\right) f(x)dx && \textit{By Inequality (10.54).} \\
&= \zeta \cdot \left(\mathbf{Pr}[\mathcal{F}_t \mid \rho_t \leq 1 + \gamma'_{\mathsf{blur}}] + \frac{1}{e^{\lambda}-1}\right) && \textit{By Law of Tot. Prob.} \\
&= \zeta \cdot \left(\mathbf{Pr}[\mathcal{F}_t \mid A_t \in \mathcal{I}_z] + \frac{1}{e^{\lambda}-1}\right) && \textit{By Definition of } A_t.
\end{aligned}$$

Recall that we observe our ball carving algorithm for τ steps. Note that it holds:

$$\mathbb{E}\left[|\mathcal{I}_z|\right] = \mathbb{E}\left[\sum_{t=1}^{\tau} \mathbb{1}_{\{A_t \in \mathcal{I}_z\}}\right] = \sum_{t=1}^{\tau} \mathbb{E}\left[\mathbb{1}_{\{A_t \in \mathcal{I}_z\}}\right] = \sum_{t=1}^{\tau} \mathbf{Pr}[A_t \in \mathcal{I}_z]$$

Further, as each endpoint of z will added to some cluster at most once, it holds:

$$\sum_{t=1}^{\tau} \mathbf{Pr}[A_t \in \mathcal{I}_z] \cdot \mathbf{Pr}[\mathcal{F}_i \mid A_t \in \mathcal{I}_z] = \sum_{t=1}^{\tau} \mathbf{Pr}[\mathcal{F}_i] \leq 1$$

Now, we compute the probability that the edge is cut in any of the τ steps. It holds:

$$\begin{aligned}
\sum_{t=1}^{\tau} \mathbf{Pr}[\mathcal{C}_t] &= \sum_{t=1}^{\tau} \mathbf{Pr}[A_t \in \mathcal{I}_z] \cdot \mathbf{Pr}[\mathcal{C}_t \mid A_t \in \mathcal{I}_z] \\
&\leq \zeta \cdot \sum_{t=1}^{\tau} \mathbf{Pr}[A_t \in \mathcal{I}_z] \cdot \left(\mathbf{Pr}[\mathcal{F}_t \mid A_t \in \mathcal{I}_z] + \frac{1}{e^{\lambda} - 1} \right) \\
&\leq \zeta \cdot \left(\sum_{t=1}^{\tau} \mathbf{Pr}[A_t \in \mathcal{I}_z] \cdot \mathbf{Pr}[\mathcal{F}_t \mid A_t \in \mathcal{I}_z] + \sum_{t=1}^{\tau} \frac{\mathbf{Pr}[A_t \in \mathcal{I}_z]}{e^{\lambda} - 1} \right) \\
&\leq \zeta \cdot \left(1 + \frac{\mathbb{E}\left[|\mathcal{I}_z|\right]}{e^{\lambda} - 1} \right)
\end{aligned}$$

Now we can wrap up the proof by considering the value of ζ. □

Now, we can plug our technical results together and see that each edge (that is sufficiently short) has only a constant probability to be endangered until step τ. Or, to put it differently, with constant probability, an edge is added to K_t if the center is close enough. It holds:

Lemma 10.23. *Let λ be the parameter of the truncated exponential distribution and assume that for some universal constants c_1, c_2 that are independent of λ, it holds:*

$$\mathbb{E}\left[|\mathcal{I}_z|\right] \leq c_1 \cdot e^{(\lambda + c_2) \cdot \left(1 + \gamma'_{\mathsf{blur}}\right)} \tag{10.55}$$

Then, for $\gamma_z \leq \gamma'_{\mathsf{blur}}$, it holds:

$$\sum_{t=0}^{\tau} \mathbf{Pr}[\mathcal{C}_t] \in O(1) \tag{10.56}$$

Proof. By Lemma 10.22, it holds

$$\sum_{t=0}^{\tau} \mathbf{Pr}[\mathcal{C}_t] \leq \left(1 - e^{-\lambda(\gamma'_{\mathsf{blur}} + \gamma_z)}\right) e^{\lambda \gamma'_{\mathsf{blur}}} \left(1 + \frac{\mathbb{E}\left[|\mathcal{I}_z|\right]}{e^{\lambda} - 1}\right) \tag{10.57}$$

Focusing on the last term, we see that it holds:

$$\left(1 + \frac{\mathbb{E}\left[|\mathcal{I}_z|\right]}{e^{\lambda} - 1}\right) \leq 1 + \frac{c_1 \cdot e^{(\lambda + c_2)(1 + \gamma'_{\mathsf{blur}})}}{e^{\lambda} - 1} = 1 + \frac{c_1 \cdot e^{\lambda(1 + \gamma'_{\mathsf{blur}}) + c_2 + c_2 \gamma'_{\mathsf{blur}}}}{e^{\lambda} - 1} \tag{10.58}$$

Recall that it holds $\gamma'_{\textsf{blur}} \leq 1$ and we can further simplify the formula as follows:

$$\left(1 + \frac{\mathbb{E}\left[|\mathcal{I}_z|\right]}{e^\lambda - 1}\right) \leq 1 + \frac{e^\lambda c_1 e^{3c_2}}{e^\lambda - 1} e^{\lambda \gamma'_{\textsf{blur}}} \tag{10.59}$$

One can easily verify that that it holds $\frac{e^\lambda}{e^\lambda - 1} \leq 2$ for all $\lambda \geq 1$. To wrap things up, we define $c_3 := 2 \cdot c_1 \cdot e^{3c_2}$ and see that it holds:

$$\left(1 + \frac{\mathbb{E}\left[|\mathcal{I}_z|\right]}{e^\lambda - 1}\right) \leq 1 + 2c_1 e^{3c_2} e^{\lambda(\gamma'_{\textsf{blur}})} \leq 1 + c_3 e^{\lambda(\gamma'_{\textsf{blur}})} \tag{10.60}$$

Note that c_3 only depends on c_1 and c_2 and is independent of $\gamma'_{\textsf{blur}} \leq 1$ and $\lambda \geq 1$ as long as they are in their given bounds. Moving on, we need some more algebra to properly bound our expression. To this end, we prove the following claim:

Claim 18. *For each constant $c \geq 0$ and $x \in [0, 1/2]$, it holds:*

$$e^x(1 - e^{-2x})(1 + c \cdot e^x) \leq c \cdot 10 \cdot x \tag{10.61}$$

Proof. We first reorder the l.h.s of the formula and see:

$$\begin{aligned} e^x(1 - e^{-2x})(1 + c \cdot e^x) &= e^x(1 + ce^x - e^{-2x} - ce^{-x}) && (10.62) \\ &= e^x(\underbrace{1 - e^{-2x}}_{(*)} + c\underbrace{(e^x - e^{-x})}_{(**)}) && (10.63) \end{aligned}$$

One can easily verify that for $x \leq 1$, it holds $e^{-x} \geq 1 - x$ as this is a well-known inequality. Thus, for the first part of the formula, it holds

$$(*) = 1 - e^{-2x} \leq 1 - (1 - 2x) = 2x \tag{10.64}$$

Further, it holds $e^x \leq 1 + x + x^2$ for $x \leq 1$ and thus, we have:

$$(**) = e^x - e^{-x} \leq (1 + x + x^2) - (1 - x) = 2x + x^2 \leq 3x \tag{10.65}$$

Plugging both inequalities back into our formula gives us the following:

$$\begin{aligned} e^x(1 - e^{-2x})(1 + c \cdot e^x) &\leq (1 + x + x^2)(2x + 3cx) = (1 + x + x^2)(x(3c + 2)) && (10.66) \\ &\leq (1 + x + x^2)5cx = 5cx + 5cx^2 + 5cx^3 && (10.67) \\ &\leq 10cx && (10.68) \end{aligned}$$

We used that $x \leq \frac{1}{2}$ in the last inequality. This proves the claim. □

Now, we can finalize the proof. By choosing both $\gamma, \epsilon \in O\left(\frac{1}{\lambda}\right)$, for a small enough hidden constant, we can ensure that the exponent $2\lambda(\gamma'_{\mathsf{blur}})$ is always smaller than $\frac{1}{2}$. Using our claim, we get that

$$\begin{aligned}
\sum_{t=0}^{\tau} \mathbf{Pr}[\mathcal{C}_t] &\leq \left(1 - e^{-\lambda(\gamma'_{\mathsf{blur}} + \gamma_z)}\right) e^{\lambda\gamma'_{\mathsf{blur}}} \left(1 + \frac{\mathbb{E}\left[|\mathcal{I}_z|\right]}{e^{\lambda} - 1}\right) \\
\text{As } \gamma_z \leq \gamma'_{\mathsf{blur}}\text{:} & \\
&\leq \left(1 - e^{-\lambda(2\gamma_z)}\right) e^{\lambda\gamma'_{\mathsf{blur}}} \left(1 + \frac{\mathbb{E}\left[|\mathcal{I}_z|\right]}{e^{\lambda} - 1}\right) \\
\text{By Inequality 10.60:} & \\
&\leq e^{\lambda\gamma'_{\mathsf{blur}}} \left(1 - e^{-2\lambda\gamma'_{\mathsf{blur}}}\right) \left(1 + c_3 e^{\lambda\gamma'_{\mathsf{blur}}}\right) \\
\text{By Claim 18:} & \\
&\leq 10 c_3 \lambda \gamma'_{\mathsf{blur}} \\
\text{As } \gamma'_{\mathsf{blur}} \leq \frac{1}{2 \cdot c_{\mathsf{blur}} \cdot \lambda}\text{:} & \\
&\leq \frac{5c_3}{c_{\mathsf{blur}}}
\end{aligned}$$

Thus, the lemma follows as c_{blur} is a constant larger than one. $\square$

Finally, we can prove Lemma 10.21.

Proof of Lemma 10.21. Let $\mathsf{Cut}_z^{(\tau)}$ be the event that edge z by some cluster K_t is cut **until** step τ. Further, let Cut_z^t be the event that edge z is cut **in** step t. Using all of our arguments and insights, we conclude:

$$\begin{aligned}
\mathbf{Pr}\left[\mathsf{Cut}_z^{(\tau)}\right] &= \mathbf{Pr}\left[\bigcup_{t=0}^{\tau} \mathsf{Cut}_z^t\right] \leq \sum_{t=0}^{\tau} \mathbf{Pr}\left[\mathsf{Cut}_z^t\right] && \textit{By Union Bound.} \\
&= \sum_{t=0}^{\tau} \mathbf{Pr}[\mathcal{C}_t] \cdot \mathbf{Pr}\left[\mathsf{Cut}_z^t \mid \mathcal{C}_t\right] + \mathbf{Pr}\left[\overline{\mathcal{C}_t}\right] \cdot \mathbf{Pr}\left[\mathsf{Cut}_z^t \mid \overline{\mathcal{C}_t}\right] && \textit{By Law of Tot. Prob.} \\
&\leq \sum_{t=0}^{\tau} \mathbf{Pr}[\mathcal{C}_t] \cdot O\left(\frac{c_{\mathsf{blur}} \cdot \lambda \cdot \ell_z}{\mathcal{D}}\right) + \sum_{t=0}^{\tau} \alpha(G_{t-1}) \cdot \frac{\ell_z}{\mathcal{D}_{BC}} && \textit{By Ineq. } (10.44) \textit{ and } (10.45). \\
&\leq O(1) \cdot O\left(\frac{c_{\mathsf{blur}} \cdot \lambda \cdot \ell_z}{\mathcal{D}}\right) + \sum_{t=0}^{\tau} \alpha(G_{t-1}) \cdot \frac{\ell_z}{\mathcal{D}_{BC}} && \textit{By Lemma 10.23.} \\
&\leq O\left(\frac{c_{\mathsf{blur}} \cdot \lambda \cdot \ell_z}{\mathcal{D}}\right) + \frac{c_{\mathsf{ldd}} \cdot \lambda \cdot \ell_z}{\mathcal{D}_{BC}} && \textit{By } (\textbf{Property 2.b}). \\
&= O\left(\frac{(c_{\mathsf{blur}} + c_{\mathsf{ldd}}) \cdot \lambda \cdot \ell_z}{\mathcal{D}}\right)
\end{aligned}$$

This was to be shown. $\square$

Complexity Finally, we analyse the time complexity, which is straightforward.

Lemma 10.24 (Complexity). *Each iteration of the algorithm can be implemented with $\tilde{O}(k)$ approximate* SetSSP *computations with parameter $\epsilon \in \Omega(1/\log^2 n)$ and minor aggregations, w.h.p.*

Proof. For the complexity, consider a fixed iteration. In the following, we summarize minor aggregations and approximate SSSP with parameter $\epsilon \in O(1/\log^2 n)$ simply as *operations*. In Step 1, we compute an LDD using the algorithm from Theorem. It requires $\tilde{O}(1)$ operations. In Step 2, we compute a weak separator using the algorithm from Theorem. It requires $\tilde{O}(k)$ operations. In Step 3, we execute a SetSSP from the separator. This clearly requires one operation. In Step 4, we execute blurry ball growing from Lemma 10.7. It requires $\tilde{O}(1)$ operations. Finally, Step 5 is purely local. This proves the lemma as there are at most $O(\log n)$ iterations, w.h.p. □

10.5.2 The Refinement Phase (Proof of Lemma 10.16)

In this section, we prove Lemma 10.16 and show that a backbone clustering can easily be extended to clustering with very favorable properties. To be precise, we show the following lemma:

Lemma 10.16. *Suppose we have an algorithm $\mathcal{A}$ that creates a (α, β, κ)-backbone clustering with pseudo-diameter $\mathcal{D}_{BC}$ of a weighted graph G. Then, there is an algorithm that creates a series of disjoint clusters $\mathcal{K} := K_1, \dots, K_N$ with $K_i := (V_i, E_i)$. Futher, it holds:*

1. ***Strong Diameter:*** *Each cluster $K_i \in \mathcal{K}$ has a strong diameter of at most $16 \cdot \mathcal{D}_{BC}$.*
2. ***Low Edge-Cutting Probability:*** *Each edge $z := \{v, w\} \in E$ of length ℓ_e is cut between clusters with probability (at most) $O\left(\frac{\ell_z \cdot (\alpha + \log \kappa\beta)}{\mathcal{D}}\right)$.*
3. ***High Clustering Probability:*** *Each node $v \in V$ is added to some cluster $K_i \in \mathcal{K}$ with probability at least $1/2$.*

If each path in the backbone is an **exact** *shortest path in its cluster, the length of the paths does affect the quality and the cutting probability improves to $O\left(\frac{\ell_z \cdot (\alpha + \log \kappa\beta)}{\mathcal{D}}\right)$.*

The algorithm requires one application of $\mathcal{A}$, $\tilde{O}(\kappa)$ minor aggregations, and $\tilde{O(1)}$ approximate SetSSP *computations with $\epsilon \in O(1/\log^2 n)$.*

We begin by proving a helpful auxiliary lemma. We show that we can efficiently compute so-called δ-nets on paths with few minor aggregations. In a δ-net, we mark a set of nodes on the path such that each node on the path has a marked node in distance δ and two marked nodes have distance at least δ.. These nets can be seen as a generalization of the Maximal Independent Set (MIS) or Ruling Sets for arbitrary distances and are formally defined as follows:

Definition 10.6 (δ-nets). *Let V be a set of nodes/points and let $d(\cdot, \cdot) : V^2 \to \mathbb{R}$ be a distance metric on these nodes/points. Then, we define a δ-net of V as a set $\mathcal{N} \subseteq V$, s.t., it holds:*

- *For each node/point $v \in V$, there is a node/point $p \in \mathcal{N}$ with $d(v, p) \leq \delta$.*
- *For two nodes/points $p_1, p_2 \in \mathcal{N}$, it holds $d(p_1, p_2) > \delta$.*

We are particularly interested in δ-nets of paths. In a sequential model, these nets can trivially be constructed by greedy algorithm that iterates over the nodes path. In the following, we sketch an algorithm that efficiently computes these nets in our model. The idea behind the algorithm is to build distance classes of length $\Theta(\delta)$ and mark one node per class. Note that the distance between net points is w.r.t. to the path, and they could be closer to each other when considering all paths in G (which will be important later).

Lemma 10.25. *Let $G = (V, E, \ell)$ be a weighted graph. Consider a set of l (not necessarily shortest) paths of length $\widetilde{O}(\mathcal{D})$ and let $\epsilon > 0$ be a parameter. Then, we can compute δ-net with $\delta = \epsilon\mathcal{D}$ for all paths in $\widetilde{O}(\epsilon^{-1} \cdot l)$ minor aggregations.*

Proof. We compute the net points on all paths sequentially. In the following, we focus on a single path P and show that the net points can be computed with $\widetilde{O}(1)$ minor aggregations. Together with the fact that there are l paths, this proves the lemma. We assume that w.l.o.g. each node knows whether it is the first or last node on a path. First, all nodes compute their distance to the first node on the path (if they do not already know it). Since the subgraphs consist of single paths, this can be done using the AncestorSum primitive. We simply need to sum up all distances on the unique path to the first/last node. Note that the distances are *exact*. Next, the last node on the path broadcasts its *distance label*, i.e., the length of the path, to all nodes on the path. Denote this distance as $\mathcal{D}_P$ in the following. Each node $v \in P$ now locally computes the smallest integer $i \in [1, \frac{\mathcal{D}_P}{\delta}]$, s.t., it holds $d_P(v, v_1) \leq i \cdot \delta$. We say that v is in *distance class* i. Each node now exchanges its distance class with its neighbors. This can be done in a single local aggregation as we can encode the distance class in $O(\log(nW))$ bits. Finally, all nodes with an *even* distance class and a neighbor in a *lower* distance class locally declare themselves net points. As a result of this procedure, the distance between a node and the next net point is at most 2δ and the distance between two net points is at least 2δ. This proves the lemma for $\delta = \epsilon/2$. □

Given these preliminaries, the high-level idea is to create a net on each backbone path and then use the resulting net nodes as a center for the algorithm from Theorem 10. The algorithm works in three steps.

(Step 1) Create a Backbone Clustering: Execute the black-box algorithm $\mathcal{A}$ to obtain a (α, β, κ)-backbone clustering $\mathcal{K} = K_1, \dots K_N$ with pseudo-diamter $\mathcal{D}_{BC}$.

The following two steps are executed parallelly in each cluster $K_i \in \mathcal{K}$.

(Step 2) Create a Net on Each Backbone Path: In each of the $O(\kappa)$ paths in the backbone $\mathcal{B}_i$ of the cluster K_i, create a $\mathcal{D}_{BC}$-net. Denote the net points created by this algorithm as $\mathcal{N}_i$.

(Step 3) Build Clusters Around Net Points: Apply **one iteration of** our algorithm from Theorem 10. That means we do not reapply the main loop until all nodes are clustered; we apply it only once. We choose the previously computed net points $\mathcal{N}_i$ as centers $\mathcal{X}_i$ in each K_i. As distance parameter choose $\mathcal{D} = 2\mathcal{D}_{BC}$. This

The lemma follows directly from the correctness of the subroutines that we use. Consider a single cluster K_i created by the black box algorithm $\mathcal{A}$. First, we want to show that each node $v \in K_i$ has at least one net point $x \in \mathcal{N}_i$ in distance $\mathcal{D}$. This follows because v must be in the distance $\mathcal{D}_{BC}$ to some path, and there is a net point in the distance $\mathcal{D}_{BC}$ to the closest node on that path. This holds in either of the two setups, as it follows directly from the properties of the backbone clustering. Thus, we fulfill the **covering property** required by the Theorem 10. In the **packing property**, we need to distinguish whether the path in the back are approximate shortest paths or not. First, consider the case that we do *not* have exact shortest paths in G. Recall that we need to bound the number of net points in distance $6\mathcal{D}$ to each node as this determines the cutting probability. As the length of each path is bounded by $O(\mathcal{D}\beta)$, we obtain $O(\beta)$ net points per path and $O(\kappa\beta)$ net points in total as we have κ paths. Clearly, each node can have at most $O(k\beta)$ net points at *any* distance. Thus, if we run the algorithm from Theorem with parameter $\tau \in O(k\beta)$ and receive clusters with strong diameter $8\mathcal{D} = 8 \cdot (2\mathcal{D}_{BC}) = 16\mathcal{D}_{BC}$. This proves the proclaimed diameter bound from Lemma. Further, each node is contained in a cluster with probability at least 1/2. This follows directly from Lemma in the analysis of Theorem 10. Thereby, we showed the clustering probability of 1/2. It remains to prove the cutting probability. An edge is cut with probability $O(\frac{\alpha\ell_z}{\mathcal{D}})$ by the (α, β, κ)-backbone clustering in Step 1 and with probability $O\left(\frac{(\log(\kappa\beta))\ell_z}{\mathcal{D}}\right)$ in Step 3. The former follows from the definition of the backbone clustering, and the latter follows from Lemma 10.10 in the analysis of Theorem 10. By the union bound, the probability of $O\left(\frac{(\alpha+\log(\kappa\beta))\ell_z}{\mathcal{D}}\right)$ follows. Further, it can be implemented with $\tilde{O}(1)$ approximate SetSSP computations and minor aggregations.

For the other bound, suppose each path is an $(1+\beta^{-1})$-approximate shortest path in K_i. Note that we only need to consider the cutting probability because the proof of the other two properties is analogous. Consider a node v and consider all net points in the distance $6\mathcal{D}$ to v. Recall that each path has length $\beta\mathcal{D}$, and we construct a $\mathcal{D}$-net. If we compute net on approximate shortest paths **of bounded length**, they also have the following extremely useful property. It holds:

Lemma 10.26. *Let $\epsilon_p, \epsilon_s \leq 1$ be two arbitrary parameters. Let P be a $(1+\epsilon_s)$-approximate shortest path of some graph $G := (V, E, \ell)$ of length $\mathcal{D} \cdot \epsilon_s^{-1}$. Further, let $\mathcal{N}$ be a $\epsilon_p\mathcal{D}$-net on P. Then, for each $c \geq 1$, every node $v \in V$ has at most $O(c\epsilon_p^{-1})$ net nodes in distance $c\mathcal{D}$ in G.*

Proof. Let $s \in V$ be the first node on path P, i.e., the node from which the approximate shortest path P was computed. We now divide path P into distance classes with respect to the distances G. For each net node $p \in \mathcal{N}$, we say that p is in distance class i_p if $i_p \in [1, O(\epsilon_s^{-1})]$ is the biggest integer such that $d_G(s,p) \leq i \cdot \mathcal{D}$, i.e., it holds:

$$i_p := \left\lceil \frac{d_G(s,p)}{\mathcal{D}} \right\rceil \tag{10.69}$$

Note that $d_G(\cdot,\cdot)$ denotes the true distance between nodes and **not** the distance on path P. The latter may be greater by $(1+\epsilon_s)$-factor.

Define $p_1 \in \mathcal{N}$ to be the first net point (counting from s) on P that is in distance $c\mathcal{D}$ to v, i.e., it holds:

$$p_1 = \underset{p \in \mathcal{N} \cap B_G(v, c\mathcal{D})}{\arg\min} \quad d_P(p,s) \tag{10.70}$$

Note that, by this definition, all net points in $B_G(v, c\mathcal{D}$ must lie *behind* p_1 on the path P, i.e., have a greater distance to s than p_1 (with respect to path P). In the following, we will only consider these net points and do not consider the earlier net points.

Let $\mathcal{U} \subset \mathcal{N}$ contain the next $N = 10c\epsilon_p^{-1}$ net nodes of P. Denote these nodes as $p_1, p_2, \dots, p_N$. Further, as we only consider net points that are further away from s than p_1, the distance between p_1 and any $p' \notin \mathcal{U}$ is at least:

$$d_P(p_1, p') = \sum_{i=1}^{N} d_P(p_i, p_{i+1}) + d_P(p_N, p') \tag{10.71}$$

$$\geq \sum_{i=1}^{N} d_P(p_i, p_{i+1}) \geq \sum_{i=1}^{N} \epsilon_p \cdot \mathcal{D} = 10 \cdot c \cdot \mathcal{D} \tag{10.72}$$

Further, let p_1 be in distance class i_{p_1}. We now argue that each net node $p' \in \mathcal{N}$ that is not in $\mathcal{U}$ must be in distance class at least $i_{p_1} + 10c - 2$. First, we consider the definition of distance class $i_{p'}$, add a dummy distance from s to v, and rearrange. We get:

$$\begin{aligned} i_{p'} &= \left\lceil \frac{d_G(s,p')}{\mathcal{D}} \right\rceil = \left\lceil \frac{d_G(s,p')}{\mathcal{D}} \right\rceil + \left\lceil \frac{d_P(s,p')}{\mathcal{D}} \right\rceil - \left\lceil \frac{d_P(s,p')}{\mathcal{D}} \right\rceil \\ &= \underbrace{\left\lceil \frac{d_P(s,p')}{\mathcal{D}} \right\rceil}_{(*)} - \underbrace{\left(\left\lceil \frac{d_P(s,p')}{\mathcal{D}} \right\rceil - \left\lceil \frac{d_G(s,p')}{\mathcal{D}} \right\rceil \right)}_{(**)} \end{aligned}$$

Now recall that the distance between p_1 and p' **on the path** P is at least $10\mathcal{D}$ by definition as $p' \notin \mathcal{U}$. Thus, for the first term, it holds:

$$(*) = \left\lceil \frac{d_P(s,p')}{\mathcal{D}} \right\rceil \tag{10.73}$$

$$\text{As } d_P(s,p') = d_P(s,p_1) + d_P(p_1,p') : \tag{10.74}$$

$$= \left\lceil \frac{d_P(s,p_1) + d_P(p_1,p')}{\mathcal{D}} \right\rceil \tag{10.75}$$

$$\text{By Ineq. (10.72)} : \tag{10.76}$$

$$\geq \left\lceil \frac{d_P(s,p_1) + 10c\mathcal{D}}{\mathcal{D}} \right\rceil \tag{10.77}$$

$$\geq \left\lceil \frac{d_P(s,p_1)}{\mathcal{D}} \right\rceil + \left\lceil \frac{10c\mathcal{D}}{\mathcal{D}} \right\rceil - 1 \tag{10.78}$$

$$= i_u + 10c - 1 \tag{10.79}$$

For the second term, we use the fact that we chose ϵ_s to be very small. In particular, we chose it small enough that the approximation error for all nodes of the path (even the nodes close to the end) is smaller than $\mathcal{D}$. It holds:

$$\begin{aligned}(**) &:= \left\lceil \frac{d_P(s,p')}{\mathcal{D}} \right\rceil - \left\lceil \frac{d_G(s,p')}{\mathcal{D}} \right\rceil && (10.80)\\ &\leq \left\lceil \frac{d_G(s,p') + \epsilon_s \cdot d_G(s,p')}{\mathcal{D}} \right\rceil - \left\lceil \frac{d_G(s,p')}{\mathcal{D}} \right\rceil && (10.81)\\ &\leq \left\lceil \frac{\epsilon_s \cdot d_G(s,p')}{\mathcal{D}} \right\rceil && (10.82)\\ \text{As } d_G(s,p') \leq \epsilon_s^{-1} \cdot \mathcal{D}: & && (10.83)\\ &< \left\lceil \frac{\epsilon_s \cdot \epsilon_s^{-1}\mathcal{D}}{\mathcal{D}} \right\rceil < 1 && (10.84)\end{aligned}$$

Combining our formulas, we get

$$\begin{aligned} i_{p'} &= \left\lceil \frac{d_P(s,p')}{\mathcal{D}} \right\rceil - \left(\left\lceil \frac{d_P(s,p')}{\mathcal{D}} \right\rceil - \left\lceil \frac{d_G(s,p')}{\mathcal{D}} \right\rceil \right) && (10.85)\\ &\geq (i_p + 10c - 1) - 1 = i_p + 10c - 2 && (10.86)\end{aligned}$$

as claimed.

We now claim that only nodes in $\mathcal{U}$ may be close to v. Now assume for contradiction that both p_1 and $p' \notin \mathcal{U}$ are in distance $c\mathcal{D}$ to v. However, this would imply that the actual distance between p and p' is at most $c\mathcal{D}$ by the triangle inequality. It holds:

$$d(v,p') \leq c\mathcal{D} \qquad (10.87)$$

Now we consider the path from s to p'. By *not* taking the path $\mathcal{P}$ from s to p', but instead the path via p_1 and v, we see that:

$$\begin{aligned} i_{p'} &:= \left\lceil \frac{d_G(s,p')}{\mathcal{D}} \right\rceil && (10.88)\\ &\leq \left\lceil \frac{d_G(s,p_1) + d_G(p_1,v) + d_G(v,p')}{\mathcal{D}} \right\rceil && (10.89)\\ &\leq \left\lceil \frac{d_G(s,p_1) + 2c\mathcal{D}}{\mathcal{D}} \right\rceil && (10.90)\\ &\leq i_{p_1} + 2c + 1 && (10.91)\end{aligned}$$

Therefore, it holds that

$$i_{p'} \underset{(10.91)}{\leq} i_{p_1} + 2c + 1 < i_{p_1} + 10c - 2 \underset{(10.85)}{>} i_{p'} \qquad (10.92)$$

This is a contradiction. Thus, no node $w \notin \mathcal{U}$ can be close to v, which implies that only nodes in $\mathcal{U}$ can be close. As the number nodes in $\mathcal{U}$ is bounded by $O(c\epsilon_p^{-1})$, the lemma follows. $\square$

Thus, by using Lemma 10.26 with $\epsilon_s = \beta^{-1}$ and $\epsilon_p = 1$, there are most $O(1)$ other points between p_1 and p_2. As we have κ paths, there are $O(\kappa)$ points in distance $\mathcal{D}_{BC}$ to node v. Thus, we run the algorithm with parameter $\tau \in O(\kappa)$, and we have an edge-cutting probability of only $O(\frac{\ell_z \cdot \log \kappa}{\mathcal{D}})$. Again, the union bound yields the claim.

10.6 Related Work

In the following, we review the related work for low-diameter decompositions in both sequential and distributed models. This section is structured as follows: First, in Section 10.6.1, we note many different notions of decompositions in the literature, and the research is not restricted to the low-diameter decompositions that follow Definition 10.1. Instead, they come in many shapes and forms with different guarantees for the clusters they produce. Nevertheless, different types of decompositions are often related to each other and/or their computations use similar techniques. We will give a short introduction to some commonly used notions of decompositions to give a good overview of how our algorithms perform. Then, we move on to the more specific related work and the present the state-of-the-art decomposition algorithms. We divide this into two parts. In the first part, we present decomposition algorithms in the (non-distributed) sequential model in Section 10.6.2. While for general graphs, our algorithm has (asymptotically) the same quality, these algorithms perform better than ours for restricted graph classes. However, in many cases, not by a large margin. Moreover, these algorithms show what might be achievable in the future. In the second part, we move on to more directly related works in the CONGEST model as there are no HYBRID algorithms to compare. Here, the situation is similar. We perform better than previous work for general graphs, while for restricted graphs, we perform worse with respect to decomposition quality. However, since our algorithm is tailored to weighted graphs, we are much faster here than previous approaches that might take linear time while we are always bounded by the hop diameter (which might be much smaller than n). A detailed comparison is presented in Section 10.6.3.

10.6.1 The Different Types of Decompositions

We begin with a generic definition that provides the basis for all forthcoming types of decomposition. A decomposition of a weighted graph $G := (V, E, \ell)$ with distance parameter $\mathcal{D}$ is a series of subgraphs $K_1, K_2, \dots$ of G, which we will call clusters. Each node $v \in V$ is contained in (at least) one of these clusters. Further, each cluster has a diameter of $\mathcal{D}$. More precisely, we say that the cluster has a *strong* diameter $\mathcal{D}$ if the distance within the cluster is $\mathcal{D}$. In this case, the path between the nodes only uses edges where both endpoints are contained in the cluster. Otherwise, if the distance between two nodes is only $\mathcal{D}$ if we are allowed to consider all edges of G, we say that the cluster has a *weak* diameter $\mathcal{D}$. Note that a cluster of weak diameter $\mathcal{D}$ is not necessarily connected. Given this underlying definition, we now present three types of decompositions. First, there is the deterministic counterpart to our probabilistic LDD's. Here, we do not bound the probability that a specific edge of length ℓ_z is cut, but instead — since it is deterministic — we count the overall number of edges of length ℓ_z that are cut. It holds:

Definition 10.7 (Deterministic Low-Diameter Decomposition). *A deterministic low-diameter decomposition with diameter $\mathcal{D}$ of a weighted graph $G := (V, E, \ell)$ with m edges is a partition of G into subgraphs $K_1, K_2, \dots$ of diameter $\mathcal{D}$, where each node is contained in exactly one cluster. We say that decomposition has quality α, the number of edges of length ℓ_z with endpoint in different clusters is bound by $m \cdot \frac{\alpha \cdot \ell_z}{\mathcal{D}}$.*

Note that $m \cdot \frac{\alpha \cdot \ell_z}{\mathcal{D}}$ is exactly the *expected* number of cut edges in probabilistic LDD of quality α. Thus, with constant probability, the probabilistic version roughly cuts the same number of edges. However, without further information about the specific random choices made by the corresponding algorithm, a probabilistic LDD could cut many more edges. While this can be mitigated with standard probability amplification techniques, i.e., executing the algorithm $O(\log n)$ and picking the iteration that cute the fewest edges, a deterministic LDD does not have this problem in the first place. In a deterministic LDD, we have the guarantee that no matter what happens in the execution of the algorithm, less than $m \cdot \frac{\alpha \cdot \ell_z}{\mathcal{D}}$ edges of length ℓ_z are cut. This makes these decompositions very useful when we do not have the time or resources for $O(\log n)$ repetitions. A typical example is distributed algorithms for local problems, where we aim for sublogarithmic runtimes. As a final remark before getting to the next type of decomposition, in unweighted graphs, some authors use slightly different notation when describing LDD's. While the definition above focuses on the clusters' diameter, some works emphasize the number of cut edges. That is, they define it as a decomposition that cuts $\epsilon \cdot m$ edges and creates a cluster of diameter $\mathcal{D} := O(\epsilon^{-1})$.

The next class of decompositions we consider in this section are so-called *padded* decompositions. While an LDD only gives guarantees for single edges and node pairs, in a padded decomposition, we have a guarantee that **all** nodes in a certain distance are contained in the same cluster. This makes them more versatile in many scenarios. Formally, they are defined as follows:

Definition 10.8 (Padded Decomposition). *A $(\beta, \gamma_{\max})$-padded decomposition with diameter $\mathcal{D}$ of a weighted graph $G := (V, E, \ell)$ is a partition of G into subgraphs $K_1, K_2, \dots$ of diameter $\mathcal{D}$, where each node is contained in exactly one cluster. For each $v \in V$, let $K(v)$ denote the cluster that contains v. Then, for each $\gamma \leq \gamma_{\max}$, it holds*

$$\mathbf{Pr}[B(v, \gamma\mathcal{D}) \subseteq K(v)] \geqslant e^{-\beta\gamma}. \tag{10.93}$$

Typically, the parameter $\gamma_{\max}$ is in the magnitude of C/β for some $C \geq 1$, so we get a guarantee for balls of diameter $C \cdot \mathcal{D}/\beta$ or smaller. While other values of $\gamma_{\max}$ are not forbidden by the definition, we are not aware of any work where $\gamma_{\max}$ is smaller than $O(1/\beta)$. With this in mind, note that the guarantees of a padded decomposition are provably stronger than the guarantees of a LDD, i.e., a $(\beta, C/\beta)$-padded decomposition implies a (probabilistic) LDD of quality α. To verify this, consider an edge $(v, w) \in E$ of length $\ell_z = \gamma\mathcal{D}$ and suppose we construct a padded decomposition. By definition of the padded decomposition, the ball $B(v, \gamma\mathcal{D})$ is fully contained in $K(v)$ with probability $e^{-\beta\gamma} \approx 1 - \beta\gamma = 1 - \frac{\beta \cdot \ell_z}{\mathcal{D}}$. In this case, the edge is not cut and thus, any padded decomposition is also a low-diameter decomposition.

The final type of decomposition that we introduce is *so-called* neighborhood or sparse covers (both names appear in the literature). As with padded decomposition, the goal is to create clusters, s.t., for every node $v \in V$ there is a cluster that contains the full ball $B(\gamma\mathcal{D})$. In contrast to the two previous concepts, a node can now be in more than one cluster to achieve this goal. However, the number of clusters that contain a given node should be kept small. Formally, they are defined as follows.

Definition 10.9 (Neighborhood/Sparse Cover). *A (γ, s)-neighborhood cover with diameter $\mathcal{D}$ of a weighted graph $G := (V, E, \ell)$ is a partition of G into subgraphs $K_1, K_2, \dots$ of diameter $\mathcal{D}$, where each node is contained in at most s clusters. Then, for each node $v \in V$ there is at least one cluster K that contains $B(v, \gamma\mathcal{D})$.*

The paramter s is sometimes called the *degree* of the cover, while $({}^1\!/\!_\gamma)$ is called the *diameter blowup*. It is easy to establish a connection between padded decompositions and neighborhood covers. Suppose that we have $(\beta, {}^C\!/\!_\beta)$-padded decomposition for parameters $\beta, C \geq 1$. Then, w.h.p., for every $c \leq C$ we can construct a $({}^c\!/\!_\beta, O(e^c \cdot \log n))$-neighborhood cover through $O(e^c \cdot \log n))$ executions of the padded decomposition algorithm. If we choose $\gamma = {}^c\!/\!_\beta$, the ball $B(v, \gamma\mathcal{D})$ is contained in $K(v)$ with probability at least e^{-c} after each execution. This follows directly from the definition of a padded decomposition. Thus, after $s := O(e^c \cdot \log n)$ independent executions of the padded decomposition algorithm, each node is in s clusters and, w.h.p., there is one cluster that contains $B(v, \gamma\mathcal{D})$. Therefore, the difficulty in studying neighborhood covers is finding a construction beats this simple trick.

10.6.2 Decompositions in Sequential Models

In the following, we provide an overview of seminal contributions and developments in low-diameter decompositions in the classical sequential model, focusing on minor-free and general graphs.

General Graphs Several algorithms construct LDD's with the optimal quality $O(\log n)$ for general graphs. In an often cited paper, Bartal [Bar96] establishes LDD's with quality $O(\log n)$ in general graphs with n vertices. The techniques used in that paper bear the greatest resemblance to our work: In each step, an arbitrary unclustered node v_i and a (roughly) exponentially distributed diameter $\mathcal{D}_i$ are chosen. Then, all nodes in the distance $\mathcal{D}_i$ to v_i are added to a cluster. More precisely, the paper shows that there exists a constant c such that, for any $\mathcal{D} > 0$ and $k \geq 1$, any n-vertex graph can be partitioned into partitions of diameter $\mathcal{D}$, such that pairs of vertices with a distance at most $\frac{\mathcal{D}}{ck}$ are clustered together with a probability of at least $n^{-\frac{1}{k}}$. Bartal [Bar96] further proves a lower bound of $O(\log n)$ for the approximation of the metric of any graph. This can be extend to show an LDD for a general graph must have a cutting probability of at least $O(\frac{\log n\ell_z}{\mathcal{D}})$ for every edge of length ℓ_z and clusters with diameter $\mathcal{D}$. Thus, Bartal's construction is optimal for general graphs. As mentioned earlier, there are other types of clustering for general graphs. In their seminal paper [AP90], Awerbuch and Peleg present a neighborhood cover for general graphs with stretch $4k - 1$ and degree $2kn^{1/k}$. That is, each node is in $2kn^{1/k}$ clusters of strong diameter Δ and every node's $\frac{\mathcal{D}}{4k-1}$-neighborhood is in one of these clusters. As noted by Fitsler in [Fil19], the algorithm in [AP90] creates $O(k \cdot n^{1/k})$ partitions with strong diameter $\mathcal{D}$, Suppose, we sample a single partition from [AP90] uniformly at random. Then, any edge (v, w) of length $\frac{\mathcal{D}}{4k-1}$ is preserved with probability at least $\Omega\left(\frac{1}{k} \cdot n^{-1/k}\right)$. Thus, it is also a weaker form of the padded decomposition and also weaker than an LDD as each edge is node's $\frac{\mathcal{D}}{4k-1}$-neighborhood is cut with the same probability. Recently, Fitsler [Fil19] improved this type of decomposition

K_r-free Graphs Now, we move the results for graphs that exclude a fixed minor K_r. Starting chronologically, Klein, Plotkin, and Rao [KPR93] demonstrate that every minor-free graph of the form K_r admits a weak decomposition scheme with padding parameter $O\left(r^3\right)$ for all distances $\Omega\left(1\right)$. This directly implies a weak diameter LDD with quality $O\left(r^3\right)$. Later, Fakcharoenphol and Talwar [FT03] improved the padding parameter to $O\left(r^2\right)$ with weak diameter. Finally, Abraham et al.[AGG+19a] prove that K_r minor free graphs admit weak $(O\left(r\right)$,$\Omega\left(\frac{1}{r}\right))$-padded decomposition scheme. Considering strong diameters, Abraham et al. [AGMW10] present a strong $2^{O(r)}$ LDD for K_r minor-free graphs. The state-of-the-art algorithm was presented by Filtser by building on the result of Abraham, Gavoille, Gupta, Neiman, and Talwar [AGG+19a]. Filtser showed that this also holds for strong decompositions for distance values smaller than $O(1/r)$. The currently best algorithm is an algorithm presented in [Fil19] that achieves $\alpha \in O(r)$. For graphs of bounded genus g and treewidth τ, [AGG+19a] presents algorithms with padding parameter $O(\log g)$ and $O(\log \tau)$. These algorithms are conjectured to be optimal. However, for general K_r-free graphs, an algorithm with padding $O(\log r)$ remains elusive.

10.6.3 Decompositions in CONGEST

In the following, we provide an overview of seminal contributions and developments in low-diameter decompositions in the CONGEST model. We put our focus on minor-free graphs and general graphs and compare the state-of-the-art with our technique. In a nutshell, the prior algorithms perform significantly better in unweighted graphs while our results are significantly faster in weighted graphs while still providing a highly non-trivial bounds. The main results are summarized in Table 10.1. Therin, we also provide an overview of our results as a reference.

General Graphs The landscape of LDD algorithms for general **weighted** graphs is very comprehensible. As mentioned before, Becker, Emek and Lenzen [BEL20] construct a low-diameter decomposition using blurry ball growing technique we reused for our construction to build a decomposition of quality $O(\log n)$ of the graph. The decomposition is weak but has the additional property that each edge is in at most $O(\log n)$ clusters. This places it between weak and strong decompositions as they show that for many applications, their notion is sufficient. Further, Rozhon, Elkin, Grunau and Haeupler [REGH22] make two significant improvements compared to [BEL20]. They present a decomposition with strong diameter (instead of weak), and their construction is deterministic (instead of randomized). Conversely, their quality is only $O(\log^3 n)$, i.e., their algorithms cut more edges than the algorithm of [BEL20]. Just as our algorithm, both algorithms only rely on approximate SetSSP computations with $\epsilon \in O(1/\log^3 n)$. Thus, they can be implemented on the CONGEST, PRAM, and HYBRID models of computation.

K_r-free graphs We now shift our focus to K_r-free graphs. For the following comparisons, we consider unweighted graphs that exclude K_r as a minor. Note that our algorithm also works (and was, in fact, designed for) weighted graphs. Recall that our algorithm always has a complexity of $\tilde{O}(r \cdot \mathrm{HD})$ for any goal diameter $\mathcal{D}$. However, in the unweighted case, this could be decreased to $\tilde{O}(\mathcal{D})$. As our algorithm works with any SSSP algorithm, we can use a faster one if $\mathcal{D} \leq \mathrm{HD}$. In the unweighted case, we can simply use the trivial

BFS algorithm that computes the distance to all nodes in the distance r in r rounds. For simplicity and easier comparison, we omit the factors that depend on the graph ' s degree in the following.

We will now compare our work to three existing CONGEST algorithms that construct similar clusterings. First, Levi, Medina, and Ron [LMR21] design a distributed algorithm for K_r-free graphs that computes a LDD with $\mathcal{D} = \epsilon^{-O(1)}$ that cuts $\epsilon \cdot n$ edges in $\epsilon^{-O(1)} \cdot O(\log n)$ rounds. This is not immediately comparable to our bounds, so we must look at the implications of this result. Recall that each K_r-free graph has $O(r \cdot n)$ edges. Therefore, if we cut $\epsilon \cdot n \in O\left(\frac{\epsilon}{r} \cdot |E|\right)$ edges, we cut a $O(\epsilon/r)$-fraction. The diameter of the cluster is $\mathcal{D} = \epsilon^{-O(1)} = \epsilon^{-c}$. For simplicity, we assume that $e^{-c} \geq r$ and the constant c hidden in the exponent's $O(1)$ is larger than 3. Thus, if measured in terms of $\mathcal{D}$, the term $\frac{\epsilon}{r}$ becomes

$$\frac{\epsilon}{r} = \frac{\mathcal{D}\epsilon}{\mathcal{D} \cdot r} = \frac{\epsilon^{-c}\epsilon^{c \cdot 1/c}}{\mathcal{D} \cdot r} = \frac{\epsilon^{-c(1-1/c)}}{\mathcal{D} \cdot r} = \frac{\mathcal{D}^{1-1/c}}{\mathcal{D} \cdot r}$$

By this calculation, we cut a $O\left(\frac{\mathcal{D}^{-1/c}}{r} \cdot |E|\right)$-fraction of the edges. Now assume that we pick a random edge z; the probability of picking an edge that is cut is $O\left(\frac{\mathcal{D}^{-1/c}}{r}\right)$. Given each edge's length is $\ell_z = 1$, the probability can be expressed as $O\left(\frac{\mathcal{D}^{1-1/c}\ell_z}{\mathcal{D} \cdot r}\right)$. Thus, we can view the process as LDD with quality $O(\mathcal{D}^{1-1/c}/r)$, which means that the quality declines with increasing $\mathcal{D}$. For any $\mathcal{D} \in o(\log\log n)$, this algorithm cuts fewer edges than ours, but for larger values of $\mathcal{D}$, our algorithm prevails. Similarly, for $\mathcal{D} \in \tilde{O}(1)$, the algorithm of [LMR21] is faster, but for larger values, our algorithm is faster.

Improving this result of Levi, Medina, and Ron, Chang and Su [CS22] show that a low-diameter decomposition with $\mathcal{D} = O(\epsilon^{-1})$ can be computed in $\epsilon^{-O(1)} \cdot \log^{O(1)} n$ rounds with high probability and deterministically in $\epsilon^{-O(1)} 2^{O(\sqrt{\log n \log\log n})}$ rounds in the CONGEST model for any K_r-free graph. Chang [Cha23] later improved the deterministic runtime to $O(\mathcal{D} \log^* n + \mathcal{D}^5)$. In a direct comparison, we perform worse as our algorithm would cut a $O(\epsilon \cdot \log\log n)$-fraction of the edges in this scenario. However, the comparison of the runtime is more nuanced. the algorithm of [CS22] is strictly better for small diameters $\mathcal{D} \in \tilde{O}(1)$ as it is faster, deterministic, and, as far as we can tell, has no galactic constants. On the other hand, our algorithm is asymptotically faster for large diameters (but also cuts more edges).

The techniques used in [CS22, Cha23] differ significantly from ours. They first perform a so-called expander decomposition to create clusters of high conductance. To be precise, their clusters have diameter $O(\epsilon^{-1})$ and cut an ϵ-fraction of the edges. Then, they show that each cluster has a high-degree node that can gather all edges of the cluster. This node can locally execute a sequential decomposition algorithm and inform the edges if they are cut. Thus, their quality is good as the best sequential clustering algorithm, which currently stands as $O(r)$ due to [Fil19]. Compared to that, the benefit of our approach is that a) it also works on weighted graphs and b) is model-agnostic, so we are not restricted to CONGEST. This universality is bought, however, by cutting more edges than [CS22, Cha23].

	Ref.	Quality	Type	Diameter	Runtime	Weighted	Comment
SEQUENTIAL	[KPR93]	$O\left(r^3\right)$	Padded	Weak	$\tilde{O}(\text{poly}\, n)$	✓	K_r-free
	[FT03]	$O\left(r^2\right)$	Padded	Weak	$\tilde{O}(\text{poly}\, n)$	✓	K_r-free
	[AGG+19a]	$O\left(r\right)$	Padded	Weak	$\tilde{O}(\text{poly}\, n)$	✓	K_r-free
	[AGMW10]	$O(e^r)$	LDD	Strong	$\tilde{O}(\text{poly}\, n)$	✓	K_r-free
	[AGG+19a]	$O\left(r^2\right)$	Padded	Strong	$\tilde{O}(\text{poly}\, n)$	✓	K_r-free
	[AGG+19a]	$O\left(\log \tau\right)$	Padded	Strong	$\tilde{O}(\text{poly}\, n)$	✓	Treewidth τ
	[AGG+19a]	$O\left(\log g\right)$	Padded	Strong	$\tilde{O}(\text{poly}\, n)$	✓	Genus g
	[Fil19]	$O\left(r\right)$	Padded	Strong	$\tilde{O}(\text{poly}\, n)$	✓	K_r- free
	[Bar96]	$O\left(\log n\right)$	Padded	Strong[d]			General graphs
CONGEST	[LMR21]	$O\left(\mathcal{D}^{1-1/c}/r\right)$	Det. LDD	Strong	$O\left(\mathcal{D}^{O(1)}\right)$	×	K_r-free
	[CS22]	$O\left(r\right)$	Det. LDD	Strong	$O\left(\mathcal{D}^{O(1)}\right)$	×	K_r-free
	Thm. 8 + Lem. 1.2	$O\left(\log \tau + \log\log n\right)$	LDD	Strong	$\tilde{O}(\tau\text{HD})$	✓	Treewidth τ
	Thm. 8 + Lem. 1.2	$O\left(\log\log n\right)$	LDD	Strong	$\tilde{O}(\text{HD})$	✓	Planar
	Thm. 8 + Lem. 1.3	$O\left(\log g + \log\log n\right)$	LDD	Strong	$\tilde{O}(g \cdot \text{HD})$	✓	Genus g
	Thm. 8 + Lem. 1.4	$O\left(f(r) \cdot \log\log n\right)$	LDD	Strong	$\tilde{O}(r \cdot \text{HD})$	✓	K_r-free
	[BEL20]	$O\left(\log n\right)$	LDD	Weak	$\tilde{O}(\text{HD} + \sqrt{n})$	✓	General graphs
	[REGH22]	$O\left(\log^3 n\right)$	Det. LDD	Strong	$\tilde{O}(\text{HD} + \sqrt{n})$	✓	General graphs
	Thm. 8	$O\left(\log n\right)$	LDD	Strong	$\tilde{O}(\text{HD} + \sqrt{n})$	✓	General graphs

Table 10.1: An overview of the related work on decomposition schemes for various graph families in the sequential model and LDD's for various graph families in the CONGEST model.

10.7 Conclusion & Future Work

In this chapter of the thesis, we presented two novel distributed decomposition schemes for general and restricted graphs that perform favorably in all relevant parameters and are applicable to a variety of models. We conclude this chapter by summarizing our results and identifying possible directions for future work.

For general graphs, we provided a decomposition scheme that produces decompositions with strong diameter and quality $O(\log n)$. In the CONGEST model, the algorithm requires $\tilde{O}(\text{HD} + \sqrt{n})$ rounds, w.h.p. If we ignore the polylogarithmic factors, then the runtime and quality are asymptotically optimal for randomized algorithms on general graphs. Therefore, a possible direction for future work would be to remove the hidden polylogarithmic factors. These factors are primarily caused by the black-box approximate SetSSP algorithm we invoke $O(\log n)$ times. Note that improving the logarithmic factors of this algorithm is an interesting goal on its own. Another open question that remains is if the guarantees of our algorithm can also be achieved deterministically. Or, in other words, is the *deterministic* construction of decompositions for general graphs with optimal runtime and quality? As our results are deeply connected with the properties of the (truncated) exponential distribution, this likely requires very different techniques and/or novel ideas for derandomization.

Second, we presented a decomposition scheme with quality $O(\log k + \log\log n)$ for universally k-path separable graphs, a graphs class that contains many graphs of practical relevance. Albeit having decomposition quality that is exponentially smaller than the quality for general graphs, our algorithm is *not* optimal and could be improved in future work. As the research on universally k-path separable graphs is scarce, the natural question is, what quality can one hope and/or should aim for!? To this end, recall that all universally k-path separable graphs exclude K_{4k+1} as a minor. So, again, we use the known results K_r-free graphs as a baseline to compare ourselves to. It has long been conjectured that any K_r-free graph admits that padded decomposition scheme with parameter $O(\log r)$. However, finding an algorithm that achieves this parameter for general K_r-free graphs remains elusive, even in the sequential setting. Here, the best-known quality is $O(r)$ due to Filtser [Fil19]. From this perspective, our algorithm appears to perform well. The dependence on k is only logarithmic, just as in the conjectured bound, and for most practical problem instances, the additive $O(\log\log n)$ should be extremely small. Thus, the bounds do not appear too bad if k is also reasonably small. But this first glance is misleading. For a better context, recall that K_r-free graphs constitute one of the most important classes of k-path separable graphs. To be precise, every K_r-free graph is universally $2^{O(\text{poly}\,(r))}$-separable. Thus, if applied to K_r-free graphs directly, our decompositions have a quality of only $O(\text{poly}\,(r) \cdot \log\log n)$. Even for small values of r, this term quickly becomes prohibitively large and unsuitable for practical applications. Therefore, the bounds specific to minor-free graphs provide a more reasonable frame of reference.

Given this sobering insight, we identify two possible directions for future work to improve the quality of the embeddings. First, one could try and show that all K_r-free graphs are (universally) $f(r)$-seperable for some $f(r) \in O(\text{poly}(r))$. The bound of $f(r) \in 2^{O(\text{poly}\, r)}$ proven by Abraham and Gavoille is not tight, so one could try to improve it using the arguments that do not rely on the decomposition by Seymour and Robertson and use perhaps more straightforward combinatorics. Here, the work by Ghaffari and Kuhn, which achieves a similar result for different graph properties for unweighted graphs, could be a starting point [GH21]. This direction does not (necessarily) require the conception of new algorithms. Instead, it requires new topological insights into K_r-free graphs that may be of independent interest.

Second, one could try to develop better distributed algorithms tailored to more specific graph classes. So, what can one hope for here? As mentioned in the related work, a distributed algorithm with a padding parameter $O(r)$ for unweighted K_r-free graphs exists already. More precisely, this algorithm can be adapted to always match the bounds of the best-known sequential algorithm. Thus, for unweighted graphs, any improvement in the realm of sequential algorithm immediately implies an improvement for unweighted graphs in CONGEST. However, the runtime of the corresponding technique depends on the emerging clusters' hop diameter. This is undesirable for weighted graphs as here the clusters' hop diameter can be as large as n even for *small* weighted diameters. For these graphs, we instead want the runtime to be in the magnitude of $O(\text{poly}(r\text{HD}) \cdot n^{o(1)})$, i.e., polynomially in the hop diameter and parameters implied by the excluded minor but sublinear in n. Therefore, we focus on weighted graphs and discuss how the existing sequential algorithms could be translated to the distributed and parallel setting. We focus on Abraham et al.'s work [AGG$^+$19a], which (together with the improvements made by Filtser [Fil19]) provides state-of-the-art guarantees for graphs that exclude a fixed minor, graphs of bounded Euler genus, and graphs of bounded treewidth. In the following, we sketch how our techniques could help adapt these algorithms to the distributed setting.

General K_r-free Graphs For general K_r-free graphs, simply combining Abraham et al.'s algorithm with ours seems promising. As our algorithm for k-path separable graphs is already heavily inspired by their algorithm, the general structure of the algorithms is very similar. In both algorithms, one samples a set of random paths (in a certain way) and builds clusters around these paths. For every connected component C, the algorithm of [AGG$^+$19a] keeps track of all clusters that neighbor a node in C. When creating a new cluster, the algorithm samples a set of shortest paths in C connecting all neighboring clusters. As the graph is K_r-free, one can show that this requires at most r paths. Then, the algorithms samples clusters around these paths. The central insight of Abraham et al. is that with their sampling technique, there are at most $O(2^r)$ sets of paths close to a given node $v \in V$ on expectation. Recall that we have shown in this thesis (in large part based on their arguments) that if there are at most τ sets of paths close to a given node $v \in V$ *on expectation*, then one can construct decomposition with quality $O(\log \tau)$. For $\tau \in O(2^r)$, the quality of $O(r)$ follows.

This sampling technique can easily be combined with ours without blowing up the runtime. In every step, we first sample a random path according to our algorithm and then some additional paths according to Abraham et al.'s algorithm. Thus, the main effort would be to show that the different sampling techniques do not affect each other too much. On the one hand, we must ensure that our additional operations and restrictions maintain Abraham et al. 's invariants and prove that each node is still only threatened by at most $O(2^r)$ paths. This is difficult because our algorithm uses approximate shortest paths instead of exact paths and shrinks the connected components, occasionally using another padded decomposition. On the other hand, we would need to show that we still sample a weak separator over time. However, if these changes can be carried through both the analysis of Abraham et al. and the construction of weak separators, the quality of the emerging decomposition would improve to $O(r)$.

Planar Graphs If adapting the algorithm for general K_r-free graphs turns out to be too difficult or simply impossible, it seems worthwhile to focus on planar graphs instead. As each planar graph excludes K_5 as a minor, the algorithm from for general K_r-free graphs can be used. However, the adaption to the distributed setting is much easier. Here, we can exploit that (a) exact shortest paths can be computed in $\tilde{O}(D^2)$ due to Li and Parter [LP19] and (b) we can compute proper (and not weak) separators that consist of $O(1)$ paths. However, recall that our approach requires a SetSSP algorithm that can be applied that can be applied in a divide-and-conquer fashion. It is not trivially clear the algorithm of Li and Parter supports this or can be extended to such an SetSSP algorithm, but it seems likely. Supposing that this is possible, we can consider the exact shortest paths when sampling paths and building separators (as in Abraham et al.'s algorithm) and also do not require the decomposition of the graph between steps (as in Abraham et al.'s algorithm). With these additional assumptions, the arguments made in the analysis of Abraham et al.'s algorithm are applicable in our case as well [2]. Therefore, a CONGEST algorithm with runtime $\tilde{O}(D^2)$ and quality $O(1)$ seems to follow directly or, at the very least, with little additional work. Further, developing an exact SetSSP for planar graphs likely has further application than just decompositions, which makes this a worthwhile direction for future work.

Graphs of Bounded Euler Genus The algorithm for graphs with bounded genus builds upon the algorithm for planar graphs. It works in three steps. First, it sequentially picks $O(g)$ shortest paths (in a certain way) to reduce the genus to 0. Then, it grows clusters around these paths. Finally, it applies the algorithm for planar graphs. This results in a decomposition of quality $O(\log g)$ as each node is threatened by at most $O(g)$ paths in the first step and $O(1)$ in the last.

[2]To to be precise, the process follows the definition of a *skeleton process* as defined in Definition 6 of [AGG$^+$19a]. Therefore, their analysis, particularly Lemma 7, applies directly.

In comparison, our technique only archives the quality of $O(\log g + \log\log n)$ even if we had access to an exact shortest path algorithm. This follows because we do not compute the shortest paths whose removal decreases the genus but sample random paths that intersect with them (which requires $O(g \log n)$ paths to remove them all). Thus, building an algorithm that can compute these paths would greatly improve the decomposition. Computing the paths directly requires (among other things) the computation of an embedding of the input graph G. Currently, to the best of our knowledge, no distributed algorithm achieves this in $\tilde{O}(\text{poly}\,(D))$ time for graphs of a genus greater than 0, i.e., for graphs of bounded genus that are not planar. With access to such an embedding, our techniques would likely be sufficient to create padded decomposition. Note that a distributed construction of such an embedding would be a breakthrough in its own right, with implications beyond constructing decompositions.

Graphs of Bounded Treewidth Finally, we conjecture that the algorithm for graphs of bounded treewidth τ can converted to the distributed setting by combining the techniques in this thesis with the algorithms presented by Izumi et al. in [IKNS22]. Therin, the authors present an exact shortest path algorithm and the efficient construction of separators that consist of $O(\tau)$ nodes in $\tilde{O}(\text{poly}\,\tau \cdot D)$ time, w.h.p. As we argued before for planar graphs, if their shortest path algorithm can be upgraded to a SetSSP algorithm, these two building blocks (exact SetSSP and fast construction of small separators) are enough for a fast decomposition scheme of optimal quality.

Summing up, we believe that - due to their generality- our techniques provide many opportunities to develop better decomposition algorithms in the future. Nevertheless, no matter the graph class, some work is required to adapt existing sequential algorithms. Further, as we heavily rely on black-box algorithms, advances in the computation of shortest paths will immediately improve our algorithms as well.

千里之行，始于足下.
(A path of a thousand miles begins with a single step.)

Chinese Proverb, attributed to Laozi

11 Distributed Construction of Compact Routing Schemes

IN this chapter, we finally consider the second major problem that was presented in the introduction. To be precise, we now consider the efficient construction of so-called *compact routing schemes* for general, k-path separable, and planar graphs in CONGEST and HYBRID. The results in the chapter are based on the following publication:

> Jinfeng Dou, Thorsten Götte, Henning Hillebrandt, Christian Scheideler, and Julian Werthmann. Brief announcement: Distributed construction of near-optimal compact routing schemes for planar graphs. In Rotem Oshman, Alexandre Nolin, Magnús M. Halldórsson, and Alkida Balliu, editors, *Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing, PODC 2023, Orlando, FL, USA, June 19-23, 2023*, pages 67–70. ACM, 2023

As mentioned in the introduction, *routing schemes* are distributed algorithms that manage the forwarding of data packets between network devices. Thus, developing efficient routing schemes is essential to enhance communication among multiple parties in distributed systems. More precisely, a routing scheme is defined as follows: Each node $v \in V$ gets assigned a *so-called* routing table $\mathcal{R}_v$. With the help of these tables, a node decides where to send a received message. Further, a routing scheme may assign a (short) label $\mathcal{L}_t$ to each node $t \in V$. These labels are attached to each message intended for $t \in V$. This means the nodes have additional information on the target node before sending the message. Upon receiving a message with label $\mathcal{L}_t$, the label of node $t \in V$, a node $v \in V$ checks its routing table $\mathcal{R}_v$ and decides the message's next hop. Thus, we can view a routing scheme as a function $f(\cdot)$ that, given a table $\mathcal{R}_v$ and a label $\mathcal{L}_t$, outputs a node $w \in V$ where the message is sent next. Repeatedly applying this function with label $\mathcal{L}_t$ starting at node $s \in V$ creates a path P_{st} from s to t in G. More formally, it holds:

$$P_{st} := (s = v_1, v_2, \dots, v_{l-1}, v_l = t)$$

For $i \in [2, l]$, it holds:

$$v_i = f\left(\mathcal{R}_{v_{i-1}}, \mathcal{L}_t\right)$$

$$v_i \in N_{v_{i-1}}$$

Here, $N_{v_{i-1}}$ denotes the neighbors of v_{i-1} in G.

In this chapter, we are interested in so-called *compact routing schemes* with *low stretch*. These routing schemes optimize the distance of their routing paths w.r.t. the shortest path metric of the graph $G = (V, E, \ell)$ representing the network. A trivial routing scheme could let each node store the complete network topology as the routing table. This, of course, is highly undesirable as it causes the routing tables to be prohibitively large. Not only does it strain the node's memory, but larger tables also take longer to compute as the nodes need to learn more information. Thus, an efficient routing should store as little information as possible. We call a routing scheme *compact* if it only stores as little information as possible on each node, i.e., the routing tables and labels are small, that is, strictly sublinear in the number of devices. Finally, the *stretch* measures the ratio between the distance of the routing scheme's path and the distance of the optimal path. Let $d_G(s, t)$ be the exact distance between nodes s, t in G and let $d_{P_{st}}(s, t)$ be the distance on the path P_{st} computed by the routing scheme. Then, the routing scheme has stretch $\alpha \geq 1$ if for all $s, t \in V$, it holds:

$$\frac{d_{P_{st}}(s, t)}{d_G(s, t)} \leq \alpha.$$

The challenge in constructing good compact routing schemes is finding schemes of *low* stretch and *small* tables and labels that can be computed *efficiently* (with respect to the model of computation).

Note that the introduction of labels simplifies the routing, as the nodes have additional information on the target. On the flip side, this introduces the burden of distributing the label of a node $t \in V$ to all nodes that want to communicate with t. This would not be necessary in a routing scheme without labels, in so-called *name-independent* routing. So, the legitimate question is why we bother with labeled routing when there is a seemingly preferable, more general, alternative. There are both practical and theoretical justifications for this. From a practical standpoint, there are mechanisms for coping with the fact that a node needs to know the address of the target it wants to reach. Notably, the Internet uses labeled routing, where the routing is based on IP addresses. The IP address of a web server is not its unique physical address but is computed and assigned based on its location in the network. The IP address for a specific webpage can be obtained through the so-called ***Domain Name Service***, or **DNS** for short [Wik24a]. Simply put, there are so-called DNS servers that store the IP addresses of all websites. If users want to visit, say `google.com`, they first send a message to a DNS server to learn the IP address of Google's web servers. In other words, they obtain the server's label. Then, they use this IP address to send their request to the actual server. Thus, labeled routing is frequently used in practice. Further, from a theoretical standpoint, the assumption of labels is, in fact, necessary for a *quick* calculation. It is easy to see that any routing scheme with stretch α implies α-approximate shortest paths. In CONGEST, the computation of approximate shortest paths takes at least $\Omega(\mathrm{HD})$ time. Recall that HD is the hop diameter of the input graph. Thus, the best we can hope for in CONGEST is a runtime of $O(\mathrm{HD})$ to compute the scheme. On the other hand, it was shown in [LP13] that there are graphs of hop diameter $O(\log n)$ in which we require $\Omega(n)$ time to compute a name-independent routing. Therefore, if we want fast algorithms that can efficiently (re-)compute a routing scheme, we have to consider labeled routing.

This chapter considers the distributed construction of routing schemes in the CONGEST model and the HYBRID model. Recall that our parameterization of the HYBRID model allows each node to send and receive $O(\log n)$ bits to/from $O(\log n)$ distinct nodes of the network in addition to the communication capabilities of CONGEST where we can send $O(\log n)$ bits along each edge.

Before we get to our contribution, let us first discuss what we can estimate as the bounds for stretch, size, and runtime we are aiming for. Regardless of the model, for a general graph, the best we hope for is a routing scheme of the stretch of $2k-1$ with routing tables of size $\tilde{O}(n^{1/k})$ due to Erdös' girth conjecture. It states that there are very dense graphs with $n^{1=1/k}$ with high girth $2k-1$. In the CONGEST model, such a routing scheme can be computed in $\tilde{O}(n^{\frac{1}{2}+\frac{1}{k}} + \mathrm{HD})$ time, w.h.p., due to a result by Elkin and Neimann [EN18b]. Given a lower bound of $\Omega(\mathrm{HD} + \sqrt{n})$ for approximate shortest paths on general graphs, the problem is essentially settled for general graphs in CONGEST. For restricted graphs, which are exempt from the girth conjecture, there are a $\tilde{O}(\mathrm{HD}^2)$ time algorithm for planar graphs [LP19] and a $\tilde{O}(\tau^2 \cdot \mathrm{HD} + \tau^3)$ time algorithm for graphs of treewidth τ [IKNS22]. Both these algorithms construct *exact* routing schemes with stretch 1. This begs the question of whether the techniques can be extended to larger classes of graphs. To put it more general, we ask ourselves:

> Is there a constant stretch compact routing scheme for a non-trivial restricted graph class that can be constructed in $\tilde{O}(\mathrm{HD})$ time in the CONGEST model?

In the HYBRID model, the situation is bit more nuanced. In a recent article, Kuhn and Schneider [KS22] prove that it takes $\tilde{O}(n^{1/3})$ rounds to compute exact routing schemes with labels of size $O(n^{2/3})$ on unweighted graphs in the HYBRID model and provide an algorithm that matches this. They also give **polynomial** time lower bounds of $\Omega(n^{1/f(k)})$ for routing schemes with stretch k on weighted graphs. Here, $f(k)$ is a function polynomial in k and independent of n. They also presented an algorithm that (almost) matches this bound and creates routing schemes of stretch $O(k)$ in $O\left(\frac{n^{1/k}}{\log n}\right)$[1] time. In both their lower bound and their algorithms, they assume LOCAL communication in the local communication mode of the HYBRID model. The lower bound, therefore, obviously also holds for us. Note that the lower bound graph is (again) a dense graph of high girth. Thus, even in HYBRID, there is an *unavoidable* super-logarithmic term in the time complexity for archiving constant stretch on general graphs. The paper left open how to construct routing schemes for restricted graphs and with less local communication. Therefore, we ask ourselves:

[1] The concrete bound in [KS22] is even more precise. We simplify it here for easier comparison. Further, the runtime is parameterized with the global communication capacity and can be sped up by increasing the capacity.

Is there a constant stretch compact routing scheme that can be constructed in $n^{o(1)}$ time in the HYBRID model using $O(\log n)$ local communication per edge?

Using a culmination of the techniques we presented in the previous chapters, we answer both the questions above affirmatively and present efficient routing schemes for the HYBRID and the CONGEST model. The main results of this chapter are summarized in the following theorem.

Theorem 12. *Distributed Construction of Routing Schemes*

Let $G = (V, E, \ell)$ be a weighted graph with polynomially bounded positive edge weights. Then the following three statements holds, w.h.p.:

- For any $k > 4$, we can compute routing scheme of stretch $O(k^2)$ with labels and tables of size $O(n^{1/k} \log^2 n)$ in $\tilde{O}\left(n^{4/k}\right)$ time in HYBRID.
- If G is $\tilde{O}(1)$-path separable, for any $\epsilon \leq 1$, we can compute routing scheme of stretch $O(1+\epsilon)$ with labels and tables of size $\tilde{O}\left(\epsilon^{-2}\right)$ in $\tilde{O}\left(\epsilon^{-3}\right)$ time in the HYBRID model and $\tilde{O}\left(\epsilon^{-3} \cdot \mathrm{HD}\right)$ time in CONGEST.
- If G is planar, for any $\epsilon \leq 1$, we can compute routing scheme of stretch $O(1+\epsilon)$ and size $O\left(\epsilon^{-1} \log^5 n\right)$ in $\tilde{O}\left(\epsilon^{-3}\right)$ time in the HYBRID model and $\tilde{O}\left(\epsilon^{-3} \cdot \mathrm{HD}\right)$ in CONGEST.

For all three statements, the HYBRID model has a local capacity of $\lambda \in O(\log n)$ and a global complexity of $\gamma \in O(\log^2 n)$.

These results improve upon the state-of-the-art in several ways. In the following, we discuss their concrete implications.

First, consider our HYBRID algorithm for general graphs. Our construction gets very close to the upper and lower bounds presented in [KS22]. However, in our parameterization of the HYBRID model, each node can only send $O(\log n)$ bits via edge. This is significantly less than in [KS22]. In that paper, they considered the LOCAL model for the local communication mode and used it to collect each node's $n^{\Theta(1/k)}$-neighborhood. On the flip side, our stretch is polynomially worse for every possible parameter $k \in O(\log n)$. Nevertheless, it offers some interesting insights depending on the value of k:

1. For constant stretch $k \in O(1)$, we are only off by a constant factor, namely k while essentially having the same runtime as [KS22].
2. On the other end of the spectrum, for $k = \Theta(\log n)$, we obtain a polylogatithmic runtime with polylogarithmic stretch using polylogarithmic communication. Such an algorithm was not known before. On the flip side, our stretch is worse by a $O(\log n)$ factor compared to [KS22].
3. For $k = \Theta(\sqrt{\log n})$, we arguably perform worst if compared to [KS22]. Here, we obtain a stretch of $O(\log n)$ in $2^{O(\sqrt{\log n})}$ time. This is quite long compared compared to the $\tilde{O}(1)$ time required by [KS22] to obtain this stretch.

Thus, our algorithm performs reasonably well in the edge cases that either optimize the runtime or the stretch.

We continue with the k-path separable graphs. Here, our results are comparable with the sequential algorithm of Abraham and Gavoille [AG06], which constructs labels and tables of size $O\left(k \cdot \epsilon^{-1} \log^2 n\right)$ in the sequential model. On the negative side, our construction is bigger by several polylogarithmic factors and (perhaps even worse) has an additional dependency on ϵ^{-1}. The latter is problematic for very small values of ϵ, i.e., if we wish to construct a nearly exact routing scheme. On the positive side, our construction can very efficiently be implemented in a fully distributed manner in both CONGEST and the HYBRID. This is our algorithm's biggest novelty, as it shows that efficient routing schemes can be distributed and constructed for very large classes of graphs if a small constant stretch is tolerable. As shown in **(author?)** [AG06] and [DG10], the family of k-path separable graphs not only contains planar graphs and graphs of bounded treewidth but also, in fact, any graph that does not contain a fixed sized minor K_r. For $\epsilon \in \Theta(1)$, i.e., when we consider the stretch to be an arbitrary small constant, the runtimes are (asymptotically) optimal: In CONGEST, we achieve a runtime of $\tilde{O}(\mathrm{HD})$, which is the lower bound for (approximate) SetSSP computations in CONGEST. In HYBRID, we have a runtime of $\tilde{O}(1)$. Here, the only thing left to optimize is the polylogarithmic factor hidden in the $\tilde{O}(\cdot)$-notation, as we did nothing to optimize this. For K_r-free graphs, we gain another benefit. Recall that the graphs are known to be $f(r)$-path separable for some $f(r)$ that only depends on r [AG06]. For these graphs, our construction avoids using the $n^{O(r)}$-time embedding algorithm used in [AG06]. Therefore, although we have larger labels and tables, our algorithm (asymptotically) is significantly faster than [AG06] if executed in a sequential manner. As mentioned earlier, however, this speed-up is *bought* with a worse dependency on ϵ^{-1} and $O(\log n)$.

Finally, our result for planar graphs compares favorably to the related work. On the one hand, it is faster and has smaller labels than the work of Li and Parter[LP19]. Their algorithm has a runtime of $\tilde{O}(\mathrm{HD}^2)$ and tables and labels of size $\tilde{O}(\mathrm{HD})$. On the other hand, their labeling is *exact* and has stretch 1. Thus, for graphs of low hop diameter, their result is preferable. However, if the hop diameter is large, and one is not interested in *exact* paths, we perform slightly better.

11.1 Structure of this Chapter

The remainder of this chapter is dedicated to proving the three statements in the theorem. It is structured as follows:

1. In Section 11.2 we present a meta-algorithm that constructs a compact routing scheme for a graph G using a so-called *tree cover*. A tree cover is a collection of forests, such that for every pair of nodes, there is a tree where the distance in this tree is approximately the distance between these nodes in G. The main idea is to create a routing scheme for each tree and combine the individual schemes into a complete routing scheme. The quality of the routing scheme, i.e., its stretch and its size, is directly dependent on the quality of the tree cover used as the basis. The basic idea of this construction is not new, it has been used before in the sequential and distributed setting.

   We present a novel-ish implementation of it that only relies on minor aggregations (in addition to computing the tree cover). More precisely, in Section 11.2, we assume that we have black-box access to an algorithm that constructs tree covers where each node is in t tress. Given the black-box, we show that with $\tilde{O}(t)$ minor aggregations, we can construct a routing scheme for G. The stretch depends on the tree cover.

2. In Section 11.3, as a warm-up, we show to construct a routing scheme with stretch $O(k^2)$ and labels and tables of size $\tilde{O}(n^{1/k})$ in HYBRID. Given the machinery from Section 11.2, we know that an efficient tree cover implies a good routing scheme. Thus, we present a tree cover for general graphs in that section. The construction is based on the pseudo-padded decompositions from Chapter 10, more precisely, Theorem 10 from Section 10.3.

3. In Section 11.4, we present an efficient tree cover for k-separable graphs, which leads to the $(1+\epsilon)$-stretch compact routing scheme with labels and tables of size $\tilde{O}(k \cdot \epsilon^{-2})$ that is promised by Theorem 12. The construction uses the pseudo-padded decomposition from Chapter 10 and also the weak separator from Chapter 8. The algorithm carefully combines both constructions to achieve the proclaimed bounds. Recall that both algorithms are based on a few $\tilde{O}(1)$ shortest-path computations and minor aggregations and, therefore, can be implemented efficiently if $k \in \tilde{O}(1)$.

4. We present related work in Section 11.5. In particular, we give an overview of (the construction of) compact routing schemes for several graph classes and in several models.

5. We conclude the chapter in Section 11.6 where we discuss some possible future directions.

11.2 Efficient Computation of Compact Routing Schemes Using Tree Covers

Many distributed and sequential algorithms that construct compact routing schemes (cf. [AG06, EN18b, EN16, IKNS22, LP19, Tho04]) follow the same basic pattern. The construction is roughly divided into two phases, a so-called *covering phase* and a *computation phase*. In the *covering phase*, we compute a series of subforests of $G := (V, E, \ell)$ that approximate the distances between all pairs of nodes. By *approximate*, we mean that the distance between two nodes in a tree, i.e., the shortest path in the tree, is *close* to their actual shortest path in G. Formally, we define these covers as follows:

Definition 11.1 (Tree Cover). *Consider a weighted graph $G := (V, E, \ell)$ and parameter $\epsilon > 0$. A $(1+\epsilon)$-approximate tree cover with overlap $t \geq 1$ for G is a series of t subforests $\mathfrak{T} := \{\mathcal{F}_1, \dots, \mathcal{F}_t\}$ with $\mathcal{F}_i \subseteq G$, s.t., for each pair $v, w \in V$, there is a forest $\mathcal{F}_i \in \mathfrak{T}$ with a tree $T \in \mathcal{F}_i$ with root r where*

$$\tilde{d}_T(v,w) := d_T(v,r) + d_T(r,w) \leq (1+\epsilon) \cdot d_G(v,w)$$

Note that each node $v \in V$ is in at most t trees per this definition, as it can be in at most one tree per forest. In the subsequent *computation phase*, we compute the actual labels and tables for the routing scheme based on these trees/forests. In particular, we construct an *exact* compact routing scheme for each individual tree in $\mathfrak{T}$. We then combine these labels and tables to obtain the routing scheme for G. If we want to route a message from s and t in G, we simply pick the tree with the least distance between s and t and route the message according to this tree's routing scheme. Crucially, the size of the resulting labels and tables depends on the number of trees a node is contained in. Further, the stretch only depends on how well the trees approximate the actual distances since the routing in the tree does not add additional distortion.

In this section, we will show how to construct a compact routing scheme from a given tree cover. In particular, we show the following:

Lemma 11.1 (Distributed Construction of Routing Schemes from Tree Covers). *Let $G := (V, E, \ell)$ be weighted graphs with edge weights bounded by W. Suppose that we have an $(1+\epsilon)$-approximate tree cover $\mathfrak{T} := \{\mathcal{F}_1, \ldots, \mathcal{F}_t\}$ of G with overlap t. Then, we can compute a routing scheme with stretch $(1+\epsilon)$ and routing labels and tables of size $O\left(t \cdot \log^2 nW\right)$ in $\tilde{O}(t)$ minor aggregations.*

Note that the construction does not depend on the specific algorithm that computed the tree cover and, therefore, can also be used as a black-box routing scheme construction combined with an algorithm that computes the tree cover.

On a high level, our construction mirrors the approach of Elkin and Neimann [EN19], which in turn was influenced by the seminal paper of Thorup and Zwick [TZ01]. For this, each node in a (rooted) tree needs to learn its DFS labels, i.e., the first and last visit of a DFS started at the root. Further, it needs to count the number of its descendants in the tree and share this number with its parent. The parent declares the child with most descendants to be its *heavy child*. Given this information, the label of a node v then consists of its entry and exit label and all the identifiers of all non-heavy children from the root to v. The DFS labels allow us to route *upward* in the tree until we find the subtree that contains the target and the identifier of the non-heavy children allow us to route *down* to target. The former works by checking if the target's entry label is in between the current nodes entry and exit label. The latter works by moving to the current node's heavy child per default unless one of its non-heavy children's identifier is in the target label. As there can be at most $O(\log n)$ of non-heavy children with identifiers of size $O(\log n)$ and DFS labels requires $O(\log n)$ bits, the label size is for single tree is $O(\log^2 n)$. Summing up all labels for all tree yields the desired result.

However, Elkin and Neimanns' computations were tailored to general graphs and had a runtime of $\tilde{O}(\text{HD} + \sqrt{n})$ in CONGEST. To speed up the computation, we will also use some techniques developed by Ghaffari and Zuzic for computations in trees[GZ22a], most notably the techniques described in Lemma 7.10. We use them to show that all of the operations sketched above can be implemented with a $\tilde{O}(1)$ minor aggregations for each forest of the tree cover. To apply this lemma, we will need to exploit the fact that tree cover can be decomposed into t disjoint forests. As $\mathfrak{T}$ consists of t disjoint forests per definition, we obtain the runtime of $\tilde{O}(t)$. While this may not be surprising, especially to readers familiar with the framework of [GZ22a], it is a necessary step to reach our runtime bounds.

The remainder is structured as follows:

- In Section 11.2.1, we present a *exact* compact routing scheme for trees. We will compute such a scheme for each individual tree in the tree cover.
- Then, we describe how to combine all these labels and tables into a routing scheme for the full graph G. The core idea is first to find the tree that best approximates the distance between source s and target t. Note that we do this *solely* based on information stored on the label. Then, we use the exact routing scheme for this tree to route to the target. We explain the details for this in Section 11.2.2.

- Finally, in Section 11.2.3, we prove Lemma 11.1 and show that the scheme has low stretch, uses little memory, and can be computed quickly.

11.2.1 An Exact Routing Scheme for Trees

In this section, we present an exact routing scheme for a single tree T_i. The core idea of the routing protocol is as follows. Suppose that we want to route from a node s to a node t. First, we determine the smallest subtree that contains both s and t, i.e., we route from s to the lowest common ancestor of s and t (which might even be s or t). Once at this node, we route *downward* until we reach t. Before we go into the details of how exactly we find these paths, let us first define the routing table $\mathcal{R}_v(T_i)$ and a label $\mathcal{L}_v(T_i)$ that we will use for this.

The Routing Table $\mathcal{R}_v(T_i)$ of a Single Tree T_i: As the routing table $\mathcal{R}_v(T_i)$, each node $v \in V$ in the tree stores the following items: First off, it stores its parent p_v, the root's identifier, and the distance to the root. The latter will later help us to pick the tree with the shortest distance between s and t. However, these distances serve no further purpose for the routing within the tree and will only come into play in the next section. For this, each node v stores entry and exit label a_v and b_v of a depth-first search started at the root. With these labels' help, we can find the least common ancestor of two nodes. Finally, each node stores h_v, the endpoint of the edge that leads to most descendants. We respectively call these the heavy edges and heavy children. This information will help us find the path from the least common ancestor to the target. All in all, we define the routing tables as follows:

Definition 11.2 (Exact Tree Routing Tables). *Let T_i be a subtree of graph $G := (V, E)$, then the routing table $\mathcal{R}_v(T_i)$ for exact routing in tree T_i looks as follows:*

$$\mathcal{R}_v(T_i) := \left(r^i \oplus d_v^i \oplus p_v^i \oplus a_v^i \oplus b_v^i \oplus h_v^i \right)$$

🟦 *r^i is tree T_i's root.*

🟪 *d_v^i is the distance from v to r^i in T_i.*

🟥 *p_v^i is the parent of v in T_i.*

🟩 *a_v^i is the entry label for a DFS in T_i.*

🟨 *b_v^i is the exit label for a DFS in T_i.*

🟥 *h_v^i is the heaviest child of v in T_i.*

Here, the operator $\oplus$ describes the concatenation of two bitstrings.

The Routing Label $\mathcal{L}_v(T_i)$ of a Single Tree T: Next, we get to the labels $\mathcal{L}_v(T_i)$ of each target node $t \in T_i$. Each label contains *all* non-heavy edges on the path from the root to the target and its entry label a_t of the depth first search. Again, it also stores the the distance to the root, which will only be important later.

Definition 11.3 (Exact Tree Labels). *Let T_i be a subtree of graph $G := (V, E)$, then label $\mathcal{L}_v(T_i)$ for exact routing in tree T looks as follows:*

$$\mathcal{L}_v(T_i) := \Big(r^i \oplus d_v^i \oplus a_v^i \oplus \big[l_v^i(1), \dots, l_v^i(k)\big] \Big)$$

- 🟦 *r^i is tree T_i's root.*
- 🟥 *d_v^i is the distance from v to r^i in T_i.*
- 🟩 *a_v^i is the entry label for a DFS in T_i.*
- ⬜ *$l_v^i(\cdot)$ is the endpoint of a non-heavyedge on the path from r_v^i to v.*

Here, the operator $\oplus$ describes the concatenation of two bitstrings.

The Routing Scheme of a Single Tree T_i: Now, we can get back to the routing scheme and fill in the missing details. The routing scheme's main idea is to first route to a subtree that contains the target and then take the heavy edge per default. The subtree can be determined via the depth first search label. Suppose the current node $v \in V$ has entry label a_v and exit label b_v. Then, if the target label is $a_t \notin (a_v, b_v)$, we send it *upward* to the node's parent. This continues until we reach the least common ancestor of source and target. If this node is not equal to the target, we send the message down to a child. Here, the identifiers of the non-heavy children come into play. On each node on the way down, we check whether the identifier of one of the neighbors is in the label. If so, then send the message to that neighbor. Otherwise, we take the heavy edge by default. One can easily verify that this scheme always finds the target as it follows the unique path between s and t in T_i.

Algorithm 1 ExactTreeRouting$(v, T_v, (m, t, \mathcal{L}_t))$

```
if $v = t$ then                                              ▷ $v$ is the message's target $t$.
    Deliver $m$
if $a_v \leq a_t \leq b_v$ then                              ▷ $t$ is in $v$'s subtree. Send downward.
    if $\exists w \in \{N_v \setminus p_v\} \cap \{l_t(1), \dots, l_t(k)\}$ then   ▷ One of $v$'s children appears in label.
        Send $(m, t, \mathcal{L}_t)$ to child $w$.
    else
        Send $(m, t, \mathcal{L}_t)$ to heavy child $h_v$.   ▷ $h_v$ is stored in $v$'s routing table.
else                                                         ▷ Send upward in $T$.
    Send $(m, t, \mathcal{L}_t)$ to parent $p_v$.            ▷ $p_v$ is stored in $v$'s routing table.
```

11.2.2 From (Routing on) Trees to (Routing on) Graphs

Now, we will finally construct our compact routing scheme for the full graph G. As before, we will first describe the contents of the routing and label of each node. For the routing table of node $v \in V$, we pick the union of all routing tables of all trees T_i in $\mathfrak{T}$ that contains v. Formally, these tables are defined as follows:

Definition 11.4 (Routing Tables for G). *Let $\mathfrak{T}$ be a tree cover for $G := (V, E)$, then the routing table $\mathcal{R}_v(\mathfrak{T})$ for graph G looks as follows:*

$$\mathcal{R}_v(\mathfrak{T}) := \bigoplus_{\mathcal{F}_j \in \mathfrak{T}} \bigoplus_{T_i \in \mathcal{F}_j} \mathcal{R}_v(T_i)$$

🟦 *$\mathcal{F}_j$ is a forest in $\mathfrak{T}$.*

🟥 *T_i is the tree of $\mathcal{F}_j$ with $v \in T_i$.*

🟩 *$\mathcal{R}_v(T_i)$ is the routing table from Def. 11.2.*

Here, the operator $\oplus$ describes the concatenation of two bitstrings.

Likewise, we define the node labels based on the union of all tree's labels.

Definition 11.5 (Node Labels for Routing in G). *Let $\mathfrak{T}$ be a tree cover for $G := (V, E)$, then the label $\mathcal{L}_v(\mathfrak{T})$ for routing in graph G looks as follows:*

$$\mathcal{L}_v(\mathfrak{T}) := \bigoplus_{\mathcal{F}_j \in \mathfrak{T}} \bigoplus_{T_i \in \mathcal{F}_j} \mathcal{L}_v(T_i)$$

🟦 *$\mathcal{F}_j$ is a forest in $\mathfrak{T}$.*

🟥 *T_i is a tree of $\mathcal{T}_j$ with $v \in T_i$.*

🟩 *$\mathcal{L}_v(T_i)$ is the label from Def. 11.3.*

Here, the operator $\oplus$ describes the concatenation of two bitstrings.

We will now describe the routing scheme. Given the target label $\mathcal{L}_t(\mathfrak{T})$ for t, a node s picks the tree T^* with the shortest distance to t that contains both s and t. By the construction of $\mathfrak{T}$, the distance between s and t must be smaller than $(1+\epsilon) \cdot d(s,t)$. Then, it uses the routing scheme for this specific tree to route the message. In more detail, the routing works as follows:

(Step 1) Find Common Trees: First, we iterate over all root identifiers stored in $R_s(\mathfrak{T})$ and $L_t(\mathfrak{T})$ and use them to determine the set $\mathcal{T}_{(s,t)} := \{T_i \in \mathfrak{T} \mid s,t \in T_i\}$. This set contains all trees that contain both s and t.

(Step 2) Find Tree T^* with Smallest Distortion: Recall that each routing table and each label also contains the distance to the root of each tree. We iterate over all trees $T_i \in \mathcal{T}_{(s,t)}$ and use the distance information to compute the values:

$$\tilde{d}_{T_i}(s,t) := d_{T_i}(s, r_s^i) + d_{T_i}(r_s^i, t)$$

Finally, we can determine tree $T^* := \arg\min_{T_i \in \mathcal{T}_{(s,t)}} \{\tilde{d}_{T_i}(s,t)\}$ that contains the shortest path between s and t (among all other trees from the tree cover).

(Step 3) Route in T^*: In the last step, we use the routing table $R_s(T^*)$ and $L_t(T^*)$ and route the message according to the routing protocol described in the previous section.

11.2.3 Proof of Lemma 11.1

In this section, we prove Lemma 11.1 using the properties of the tree cover and the tree operations presented in Lemma 7.10. We divide the proof into three parts. First, we show that the routing scheme indeed has a stretch $(1+\epsilon)$. Then, we prove that the size of the labels and tables are within $O(t\cdot\log^2 n)$ and $O(t\cdot\log n)$, respectively. Finally, we show that everything can be computed within $\tilde{O}(t)$ minor aggregations.

Stretch We route the message in the tree $T^* := \arg\min_{T_i \in \mathcal{T}_{(s,t)}} \{\tilde{d}_{T_i}(s,t)\}$ that contains the shortest path between s and t via some tree's root. Recall that by definition of the tree cover, there must be a tree for which it holds:

$$\tilde{d}_{T_i}(s,t) := d_{T_i}(s,r_s^i) + d_{T_i}(r_s^i,t) \leq (1+\epsilon)\cdot d_G(s,t)$$

Therefore, it holds $\tilde{d}_{T^*}(s,t) \leq (1+\epsilon)\cdot d_G(s,t)$. Further, by the triangle inequality, it holds:

$$\begin{aligned} d_{T^*}(s,t) &\leq d_{T^*}(s,r_s^*) + d_{T_i}(r_s^*,t) \\ &:= \tilde{d}_{T^*}(s,t) \leq (1+\epsilon)\cdot d_G(s,t) \end{aligned}$$

Thus, the distance between s and t is T^* is at most $(1+\epsilon)\cdot d_G(s,t)$. As the routing scheme for T^* is exact, the routing scheme has a stretch of $(1+\epsilon)$ in G as claimed.

Label and Table Size Recall that the labels and tables are the union of t labels and tables of trees. Thus, we show that the labels and tables of each tree are of size $O(\log nW)$ and $O(\log^2 n + \log nW)$, respectively. Multiplying these values by t gives the proclaimed sizes from Lemma 11.1. We begin with the tables and show:

Lemma 11.2 (Size of Routing Tables). *The tables described in Definition 11.2 can be expressed within* $O(\log(nW))$ *bits.*

Proof. The table consists of six fields of size $O(\log nW)$. Three of them, the root r^i, the parent p_v^i, and heavy child h_v^i are node identifiers. These are of size $O(\log n)$ per definition. Further, two fields a_v^i and b_v^i are the entry and exit labels of a DFS. The labels of a DFS are in the range $1,\ldots,|E|$ where $|E|$ is the number of edges. Thus, we require $O(\log|E|)$ bits to encode them. As $|E| \in O(n^2)$, this is within $O(\log n)$. Finally, there is the distance d_v^i between v and r^i. This is at most Wn and therefore requires $O(\log nW)$ bits. As each of these six items is either a node identifier or a number smaller than Wn^2, the total information sums up to $O(\log(nW))$. □

Next, we consider the labels and show:

Lemma 11.3 (Size of Routing Labels). *The labels described in Definition 11.3 can be expressed within* $O(\log^2(n) + \log(nW))$ *bits.*

Proof. First, we note that there can be at most $O(\log n)$ non-heavy edges on the path to each node. To see this, recall that the number of nodes in a non-heavy child's subtree is at most half the number of nodes in the parent's subtree. Otherwise, the child would be heavy because there cannot be two children with more than half the descendants. So, each non-heavy edge reduces the number of possible targets by half and there can be at most $O(\log n)$ of them. Thus, the total number of bits required for each label is $O(\log^2(n) + \log(nW))$ as each of the $O(\log n)$ non-heavy edges has a label of size $O(\log n)$ and all other other item can be encoded in $O(\log nW)$ bits (as argued in the proof of the previous lemma). □

This concludes the analysis of the routing scheme's size.

Complexity Finally, we consider the computation of the routing scheme for G. Again, we split our analysis into the computation of the tables and labels. As we will see, for each tree, the computation requires $\widetilde{O}(1)$ minor aggregations. Adding all these runtimes for all these trees up, yields the desired complexity. In particular, we will make heavy use of the following lemma:

Lemma 7.10 (Tree Operations, Based on [GZ22a]). *Let $F := (T_1, \dots, T_m)$ be a subforest (each edge e knows whether $e \in E(F)$ or not) of a planar graph and suppose that each tree T_i has a unique root $r_i \in V$, i.e., each node knows whether it is the root and which of its neighbors are parent or children, if any. Now consider the following three computational tasks:*

1. ***AncestorSum and SubtreeSum:*** *Suppose each node $v \in T_i$ has an $\widetilde{O}(1)$-bit private input x_v. Further, let $Anc(v)$ and $Dec(v)$ be the ancestors and descendants of v w.r.t. to r_i, including v itself. Each node computes $A(v) := \bigotimes_{w \in Anc(v)} x_w$ and $D(v) := \bigotimes_{w \in Dec(v)} x_w$.*
2. ***Path Selection:*** *Given a node $w \in T_i$, each node $v \in T_i$ learns whether it is on the unique path from r_i to w in T_i.*
3. ***Depth First Search Labels:*** *Each node $v \in T_i$ computes its unique entry and exit label of a depth first search started in r_i.*

All of these tasks can be implemented in one round of minor aggregation.

We begin with the construction of the tables. Algorithm 2 summarizes all operations that need to be executed to compute the table. Formally, we show that it holds:

Lemma 11.4. *Let $\mathfrak{T} := \{\mathcal{F}_1, \dots, \mathcal{F}_t\}$ be a tree cover with overlap t according to Definition 11.1. Then, the routing tables described in Definition 11.2 can be computed in $\widetilde{O}(t)$ rounds of minor aggregations for all trees.*

Proof. All items in the table can be efficiently computed using a few minor aggregations and the techniques presented in Lemma 7.10. The lemma requires the trees on which the aggregation is executed to be disjoint, i.e., they must be a forest. Thus, the following routine will be executed for every forest $\mathcal{F} \in \mathfrak{T}$. The identifier of the tree's root can be delivered via a simple broadcast in each tree. Further, the parent in the respective tree is already known through the construction of the tree cover. So, we need no more time to compute them. If not already known, the distance to the root can be computed through the AncestorSum primitive from Lemma

Algorithm 2 ComputeTable(T)

```
for $v \in V$ do
    $r_v \longleftarrow$ RootOf$(T)$                                  ▷ Computed via broadcast.
    $p_v \longleftarrow$ ParentOf$(v, T)$                             ▷ Part of input.
    $d_v \longleftarrow d_T(v, r_v)$                 ▷ Distance to root in $T$. Computed via AncestorSum.
    $(a_v, b_v) \longleftarrow$ DFS$(v, T)$          ▷ DFS$(v, T)$ returns entry and exit label of a DFS.
    $\omega_v \longleftarrow \sum_{w \in \mathsf{Desc}(v,T)} 1$        ▷ Descendants in $T$. Computed via SubtreeSum.
    $h_v \longleftarrow \arg\max_{w \in C_v} \{\omega_w + 1\}$   ▷ Heaviest child. Computed via local aggregation.
    $\mathcal{R}_v(T) := (r_v, d_v, p_v, a_v, b_v, h_v)$
return $(\mathcal{R}_v(T))_{v \in V}$
```

7.10. For this, each node $w \in V$ picks the distance to its parent as input $x_w = d(w, p_w)$. A depth-first search's entry and exit label can be computed directly with the subroutine promised in Lemma 7.10. Finally, the heavy edges can be determined via SubtreeSum technique presented in Lemma 7.10. Each node $w \in T_i$ simply picks $x_w = 1$ as its private input, and we chose SUM as our aggregation operator. Then, all nodes compute $D(v) = \sum_{w \in Dec(v)} x_w$ as defined in Lemma 7.10. Recall that $Dec(v)$ denotes the set of descendants of v. Now, each node knows the total number of its descendants. Finally, each node determines the maximal value $D(w)$ among its children to elect the heaviest child. In case of a tie between two or more children, the node identifier is used to break it. By Lemma 7.10, each of these operations takes $\tilde{O}(1)$ rounds of minor aggregations. Repeating it for all t forests $\mathcal{F}_1, \ldots, \mathcal{F}_t$ yields the result. □

Now, we continue with the computation of the labels. Here, it similarly holds:

Lemma 11.5. *Let $\mathfrak{T} := \{\mathcal{F}_1, \ldots, \mathcal{F}_t\}$ be a tree cover with overlap t according to Definition 11.1. Then, the labels described in Definition 11.3 can be computed in $\tilde{O}(t)$ rounds of minor aggregations.*

Proof. Note that the labels can be efficiently computed via the minor aggregation techniques we described in Lemma 7.10. Note that our previous lemma directly provides a runtime bound for root's identifier, the distance, and the depth first search labels. Thus, we focus on computing the identifiers of the non-heavy edges, which is a bit trickier. As mentioned in the computation of the routing tables, the heavy edges can be determined via SubtreeSum. Since each parent learns the identifier of their heavy child, it can also inform the respective child that it is heavy. All children that do not get such a message from their parents must therefore be the endpoints of non-heavy edges. We will now use the AncestorSum primitive from Lemma 7.10 to let all nodes learn their non-light edges on the path to the root. To this end, all endpoints of non-heavy edges $w \in T_i$ pick their identifier as their private input x_w. As the aggregation operator, we pick $\oplus$, the concatenation. Since there can be at most $O(\log n)$ non-heavy edges on the path from the root to a node v, the aggregate value that each v needs to learn is of size $O(\log^2(n))$. Thus, for each node $A(v) = \bigoplus_{w \in Anc(v)} x_w$, where $Anc(v)$ means ancestors of v, the subset of the ancestors that are not heavy children of their parents, can be determined via the AncestorSum primitive. □

Thus, together both lemma imply that $\tilde{O}(t)$ minor aggregations are required to compute routing tables and labels for all trees in a tree cover with overlap t. This concludes the analysis and this section.

11.3 Tree Covers Using Pseudo-Padded Decompositions

Given the machinery introduced in the previous section, the problem of constructing a compact routing scheme can be reduced to finding a tree cover. For general graphs with polynomially bounded edge weights, tree covers can be easily constructed by repeatedly constructing pseudo-padded decompositions from Theorem 9 in Section 10.2. Using them as a black box, we can show the following general lemma:

Lemma 11.6. *Let $G = (V, E, w)$ a graph with polynomially bounded edge weights. Then, for $k > 4$ we can construct a $O(k^2)$-approximate tree cover with overlap $O\left(n^{1/k} \log n\right)$ within $\tilde{O}\left(n^{4/k}\right)$ time in HYBRID, w.h.p.*

Proof. First, we construct a $2k-1$-spanner $G_k = (V, E_k)$ of G. For any two nodes $v, w \in V$, its holds that:

$$d_{G_k}(v, w) \leq (2k-1) \cdot d_G(v, w)$$

This takes $O(k^2)$ time using the algorithm of Baswana and Sen [BS07]. It is a CONGEST algorithm that naturally has the same runtime in HYBRID. In the following, we perform all computations on G_k instead of G.

For the remainder of this proof, let $\mathcal{D}_i := 2^i$. Fix two nodes $v, w \in V$ and suppose that $d_{G_k}(v, w) \in [\mathcal{D}_{i-1}, \mathcal{D}_i]$. We will show how to construct a tree cover that contains a tree T with $d_T(v, w) \in O(k \cdot \mathcal{D}_i)$, w.h.p. For a fixed $\mathcal{D}_i$, we do the following: We construct a pseudo padded decomposition from Theorem 9 with distance parameter

$$\mathcal{D}_i' = 160 \cdot k \cdot \mathcal{D}_i$$

and error parameter

$$\epsilon = \left(1000 \cdot n^{1/k} \cdot \log n\right)^{-1} \tag{11.1}$$

on G_k. We choose every node as a possible center, so we have $\mathcal{X} = V$. With this parameterization, the algorithm creates clusters of strong diameter $O(k \cdot \mathcal{D}_i)$. Further, each cluster contains a $(1+\epsilon)$-approximate SetSSP-tree T that is rooted in the cluster's center. Thus, if a cluster contains both v and w, there is tree T such that, it holds:

$$d_T(v, w) \leq c \cdot k \cdot \mathcal{D}_i \leq 2c \cdot k \cdot d_{G_k}(v, w) \leq 4c \cdot k^2 \cdot d_G(v, w)$$

Here, we used the assumption that $d_{G_k}(v, w) \in [\mathcal{D}_{i-1}, \mathcal{D}_i]$ and that G_k is a $(2k-1)$-spanner. Therefore, this tree has the desired properties. Thus, it remains to bound the probability that a cluster contains both v and w. To this end, let C_v be the cluster that contains v. Further, define

$$\gamma = \frac{1}{160 \cdot k} \tag{11.2}$$

Note that $\epsilon \leq \gamma \leq 1/32$. For a fixed node $v \in V$, the probability that a node $w \in B_{G_k}(v, \mathcal{D}_i)$ is in the same cluster as v can be bounded as follows:

$$\begin{aligned}
\mathbf{Pr}[B_{G_k}(v,\mathcal{D}_i) \subset C_v] &= \mathbf{Pr}[B_{G_k}(v,\gamma\mathcal{D}_i') \subset C_v] \\
\text{By Lemma 10.2}: & \\
&\geq e^{-16(\gamma+5\epsilon)\log n} - 160\epsilon \log n \\
\text{As } \epsilon \leq \gamma: & \\
&\geq e^{-80\gamma \log n} - 160\epsilon \log n \\
\text{By (11.1) and (11.2)}: & \\
&\geq e^{-\frac{\log n}{k}} - \frac{160 \log n}{1000 \cdot \log n \cdot n^{1/k}} \\
&\geq n^{-1/k} - (1/2) \cdot n^{-1/k} = (1/2) \cdot n^{-1/k}
\end{aligned}$$

Thus, after creating $O(n^{1/k} \cdot \log n)$ such decompositions, there must be one with a cluster that contains both v and w, w.h.p. We add all resulting spanning trees of the clusters to the tree cover. As we add one tree per cluster, each node is in $O(n^{1/k} \cdot \log n)$ trees and all trees can be divided into $O(n^{1/k} \cdot \log n)$ forests.

As we assume the weights of G to polynomially bounded, there are $O(\log n)$ possible values of $\mathcal{D}_i$. After repeating the construction above for all $O(\log n)$ possible values of $\mathcal{D}_i$, we obtain a suitable tree for every pair of nodes, w.h.p. Thus, we have a $O(k^2)$-approximate tree cover with overlap $O\left(n^{\frac{1}{k}} \log^2 n\right)$ as claimed.

It remains to bound the algorithm's time complexity. Each pseudo-padded decomposition requires exactly one $(1+\epsilon)$-approximate SetSSP computation that dominates the runtime. Note that G_k has an arboricity of $O(n^{\frac{1}{k}})$. This was remarked in [AHK+20a]. Therefore, we can perform a $(1+\epsilon)$-approximate SetSSP on G_k in $\tilde{O}(\epsilon^{-2} \cdot n^{\frac{1}{k}})$ time in HYBRID by simulating the PRAM algorithm from [RGH+22]. Thus, the construction of single decomposition takes $\tilde{O}(n^{3/k})$ time, w.h.p. As we require $O(n^{1/k} \cdot \log^2 n)$ decompositions, the total time complexity is $\tilde{O}(n^{4/k})$, w.h.p. This proves the lemma. □

Thus, using the tree cover from Lemma 11.6 in the construction from Lemma 11.1, we can construct the routing scheme in $\tilde{O}(n^{4/k})$ minor aggregations in HYBRID, w.h.p. Now recall that a minor aggregation in HYBRID requires $O(\log n)$ rounds. This results in a runtime of $\tilde{O}(n^{4/k})$, w.h.p. This proves the first statement from Theorem 12.

11.4 Tree Covers Using Weak Separators

In this section, we describe a construction of compact routing schemes for k-path separable graphs that only use approximate SetSSP computations and minor aggregations. We achieve this by efficiently constructing a tree cover with a small overlap. As in previous works, our key idea is to find a *small* set of nodes that intersect with many shortest paths, i.e., a *hitting set* for the shortest paths, and construct trees rooted in this set. The *size* of these hitting sets greatly affects the overlap of the resulting tree cover. For example, previous works used skeleton graphs[LP15], hopsets[EN18b], or vertex separators [IKNS22, LP19] as the basis for their constructions. We will construct a special type of separator for a k-path separable graph G that fulfills this role, namely the weak $(\mathcal{D}, \epsilon)$-separator from Chapter 8. These are defined as follows:

Definition 8.1 (Weak κ-Path $(\mathcal{D}, \epsilon)$-Separator). *Let $G := (V, E, w)$ be a weighted graph, $\mathcal{D} > 1$ be an arbitrary distance parameter, and $\epsilon > 0$ be an approximation parameter. Then, we call the set $S(\mathcal{D}, \epsilon) := (\mathcal{P}_1, \dots, \mathcal{P}_\kappa)$ with $\mathcal{P}_i := (P_i, B_i)$ a weak κ-path separator, if it holds:*

1. *Each $P_i \in \mathcal{P}_i$ is a (approximate) shortest path in $G \setminus \bigcup_{j=1}^{i-1} \mathcal{P}_j$ of length at most $4\mathcal{D}$.*
2. *Each $B_i \subseteq B_G(P_i, \epsilon\mathcal{D})$ is a set of nodes surrounding path P_i.*
3. *For all $v \in (V \setminus \bigcup_{j=1}^{\kappa} \mathcal{P}_i)$ it holds $|B_{G \setminus S}(v, \mathcal{D})| \leq (7/8) \cdot n$.*

In the following, we assume we can access a weak κ-path $(\mathcal{D}, \epsilon)-$ separator construction algorithm as a black-box. In particular, we assume that we have access to an oracle that constructs these separators:

Definition 11.6 (Separator Oracle $\mathcal{O}_\kappa^{\mathcal{D},\epsilon}$). *Let $G = (V, E, \ell)$ be a weighted graph and let $\mathcal{C}_1, \dots, \mathcal{C}_N$ be a series of N disjoint connected subgraphs of G. Further, let $\mathcal{D} \geq 1$ and $\epsilon \leq 1$ be parameters. Then, an application of the oracle $\mathcal{O}_\kappa^{\mathcal{D},\epsilon}$ computes a κ-path weak $(\mathcal{D}, \epsilon)$-separator in every $\mathcal{C}_1, \dots, \mathcal{C}_N$.*

This allows us to use different algorithms to construct the weak path separators and also makes our construction open to future improvements. We will use it to construct a $(1 + \epsilon)$-approximate tree cover with low overlap $\tilde{O}(\epsilon^{-1} \cdot \kappa \cdot \log W)$.

On a high level, our algorithm is a distributed and parallel implementation of Thorup's algorithm for tree covers [Tho04] that was also used by Abraham and Gavoille for k-path separable graphs [AG06]. However, our weak κ-path $(\mathcal{D}, \epsilon)$-separator differs from the (classical) k-path separator used in these algorithms. Not only does it contain more paths and additional nodes, but the paths are only approximate shortest paths, and when removed, they do not necessarily create disjoint connected subsets. For these reasons, simply replacing the separators does not work. In addition to the weak path separators, we will also need our generic clustering algorithms to resize the graphs. To be precise, our construction is based on so-called $(\epsilon, \mathcal{D})$-additive tree covers, which are formally defined as follows.

Lemma 11.7 ($(\epsilon, \mathcal{D})$-additive Tree Cover). *Let $\mathcal{D}, \epsilon \geq 0$ be parameters. An $(\epsilon, \mathcal{D})$-additive tree cover with overlap $t \geq 1$ for a graph G is a series of rooted trees $\mathcal{T}(\epsilon, \mathcal{D}) := (T_1, T_2, \dots)$, s.t. it holds:*

1. *Each node $v \in V$ is in at most t trees.*
2. *For each pair $v, w \in V$ with $d(v, w) < 2\mathcal{D}$, there is a tree $T \in \mathcal{T}$ with*

$$d_T(v, w) \leq (1 + \epsilon) \cdot d_G(v, w) + \epsilon\mathcal{D}$$

This section's main result is that we can compute a polylogarithmic-size $(\epsilon, \mathcal{D})$-additive tree cover for any distance $\mathcal{D}$ and error parameter ϵ. Given the techniques we developed in the previous chapters, we show the following lemma:

Lemma 11.8 (Distributed Tree Covers for k-Path Separable Graphs). *Consider a weighted k-path separable graph $G := (V, E, \ell)$. Suppose that we have black-box access to an oracle $\mathcal{O}_{\kappa}^{\mathcal{D},\epsilon}$. Then, an $(\epsilon, \mathcal{D})$-additive tree cover with overlap $O(\kappa \cdot \epsilon^{-1} \cdot \log^4 n)$ can be computed with $O(\log^2 n)$ applications of $\mathcal{O}_{\kappa}^{\mathcal{D}',\epsilon'}$ with $\epsilon' \in \Omega\left(\epsilon/\log^2 n\right)$ and $\mathcal{D}' \in O\left(\mathcal{D} \cdot \log^2 n\right)$.*

Further, the algorithm requires $\tilde{O}(\kappa \cdot \epsilon^{-1})$ minor aggregations and $\tilde{O}(\kappa \cdot \epsilon^{-1})$ $(1+\epsilon'')$-approximate SetSSP computations where $\epsilon'' \in \Omega\left(\epsilon/\log^3 n\right)$.

For a fixed $\mathcal{D}_i$, the algorithm works as follows: We then apply our pseudo-padded decomposition algorithm promised by Theorem 9 with parameter $\mathcal{D}_i' := O(\mathcal{D}_i \cdot \log^2 n)$ and $\epsilon \in O\left(1/\log^2 n\right)$ to the graph. Recall that this algorithm is implemented solely via $O(1)$ approximate shortest-path computations. The algorithm decomposes the graph into connected subgraphs of diameter at most $\mathcal{D}_i'$ and preserves the $2 \cdot \mathcal{D}_i$-neighborhood of all but a $O(1/\log n)$-fraction of nodes. In each connected component, we compute a weak $(\mathcal{D}_i', \epsilon)$-separator S according to Definition 8.1. Using the black-box algorithm $\mathcal{O}_{\kappa}^{\mathcal{D},\epsilon}$, this results in a separator $S(\mathcal{D}, \epsilon) := (\mathcal{P}_1, \dots, \mathcal{P}_{\kappa})$ that consists of κ approximate shortest paths $P_1, \dots, P_{\kappa}$ (and some nodes close to them). Then, we construct a set of so-called *portals* on each path P_l. These are a subset of P_l's nodes, s.t., it holds that (1) each node on P_l is in the distance at most $\epsilon\mathcal{D}_i$ to a portal, and (2) two portals are in the distance at least $\epsilon\mathcal{D}_i$ from each other. In other words, for each path, we compute a $O(\epsilon \cdot \mathcal{D}_i)$-net. We can use the algorithm from Lemma 10.25 for this. As stated in the lemma, the algorithm only uses minor aggregations and can, therefore, be executed in parallel for a single path per connected component. As there are κ paths per component, computing all nets for all paths requires $\tilde{O}(\kappa)$ minor aggregations. Finally, we sequentially compute $(1+\epsilon)$-approximate shortest-path trees rooted in previously computed net points. These will be the actual trees of the tree cover. As the length of the random paths is bounded by $O(\mathcal{D}_i')$, there cannot be too many net points per path, i.e., at most $O(\epsilon^{-1} \cdot \log^2 n)$ many. All these $O(\epsilon^{-1} \cdot \log^2 n)$ trees of the κ paths will be added to the tree cover. Finally, we recursively apply this algorithm to the remaining subgraph again until the graph is empty. As we remove a weak $\mathcal{D}_i'$-separator from every component, the subgraph that results from the decomposition at the beginning of the algorithm contains at most $(7/8) \cdot n$ nodes. As k-separable graphs are closed under minor-taking [DG10], the resulting subgraphs are k-path separable as well. Therefore, it is easy to verify that after $O(\log n)$ such recursive applications of this routine, the graph is empty, w.h.p. Thus, it creates $O(\epsilon^{-1} \cdot \kappa \cdot \log^3 n)$ forests in total. Now consider a pair of nodes $v, w \in V$ in the distance at most $2 \cdot \mathcal{D}_i$. In the analysis, we will show that the algorithm above will, with *constant* probability, construct a tree that approximates their distance. Thus, if we independently repeat this construction $O(\log n)$ times, we construct a suitable tree, w.h.p. For a more detailed description of the algorithm and the concrete choice of the parameters that we will hide in the $\tilde{O}$-notation, we refer to Subsection 11.4.

Lemma 11.9 (From Additive To Full Tree Covers). *Let $\mathcal{A}^{\mathcal{D},\epsilon}$ be an algorithm that creates and $(\epsilon, \mathcal{D})$-additive tree cover with overlap t for a graph $G := (V, E, \ell)$ with edge weights bounded by W. Then, $d^* := \log nW$ executions $\mathcal{A}^{\mathcal{D}_1,\epsilon'}, \dots, \mathcal{A}^{\mathcal{D}_{d^*},\epsilon'}$ with $\mathcal{D}_i = 2^i$ and $\epsilon' = \epsilon/6$ create $(1+\epsilon)$-approximate tree cover with overlap $O(t \cdot \log nW)$.*

The high-level idea behind the construction is to consider the distance scales $\mathcal{D}_i := 2^i$ for $i \in [1, \lceil \log nW \rceil]$ separately. For each of these scales, we compute a separate tree cover that approximates the distance between all pairs of nodes in the distance at most $2\mathcal{D}_i$. The resulting collection of these $O(\log nW)$ tree covers $\mathfrak{T} := \{\mathcal{T}(\epsilon, 1), \ldots, \mathcal{T}(\epsilon, nW)\}$ is the desired *tree cover*. It is easy to see that for each pair of nodes, such a $\mathfrak{T}$ contains a tree that approximates the distance between them. Repeating this for all $O(\log nW)$ distance scales produces a tree cover with overlap $\tilde{O}(\epsilon^{-1}\kappa)$.

Equipped with this lemma, we can now show the latter two statements of Theorem 12. In particular, we use the two separator constructions from Chapter 8 to implement the oracle $\mathcal{O}_{\mathbf{Sep}}^{\mathcal{D},\epsilon}$. In the following, we consider the result for the k-path separable graphs and planar graphs separately.

k-Path Separable Graphs By Lemma 6, we can construct a weak κ-path $(\mathcal{D}, \epsilon)$-separator with $O(k \cdot \epsilon^{-1} \cdot \log n)$ paths in both HYBRID and CONGEST. Thus, by using the separator construction from Lemma 6 in Lemma 11.8 we can construct $(\mathcal{D}, \epsilon)$-additive tree covers with overlap $O(k \cdot \epsilon^{-1} \cdot \log^2 n)$. Plugging these additive separators into Lemma 11.9 gives us a $(1+\epsilon)$-approximate tree cover with overlap $O(k \cdot \epsilon^{-1} \cdot \log^6 n)$. Building a routing scheme based on this tree cover with 11.1 implies the second statement of Theorem 12. To be precise, the labels and tables of the routing scheme are of size:

$$\underbrace{O\left(k\epsilon^{-1}\log^2 n\right)}_{\substack{\textit{Paths in Separator}\\(\textit{Lemma } 6)}} \cdot \underbrace{O\left(\log(nW)\right)}_{\textit{Distance Scales}} \cdot \underbrace{O\left((\epsilon^{-1}\log^4(n)\right)}_{\substack{\textit{Trees per Distance Scale}\\(\textit{Lemma } 11.8)}} \cdot \underbrace{O\left((\log^2(nW))\right)}_{\substack{\textit{Bits per Tree Label}\\(\textit{Lemma } 11.1)}} = O\left(k\epsilon^{-2}\log^9(nW)\right)$$

Further, the required number of minor aggregations and SetSSP computations to construct the separator is $O(k \cdot \epsilon^{-1} \cdot \log n)$. Thus, in each application of Lemma 11.8, we require $O(k \cdot \epsilon^{-1} \cdot \log^3 n)$ minor aggregations to compute the separators. Further, the Lemma requires $O(k \cdot \epsilon^{-1} \log n) \cdot \tilde{O}(\epsilon^{-1})$ further minor aggregations and approximate SetSSP computations. We conclude that the total number of minor aggregations and approximate SetSSP computations to compute the routing scheme is therefore $\tilde{O}\left(k \cdot \epsilon^{-2}\right)$.

Finally recall that $(1+\epsilon)$-approximate SetSSP computations and minor aggregations, w.h.p., require $\tilde{O}\left(\epsilon^{-2}\right)$ time in HYBRID and $\tilde{O}\left(k \cdot \epsilon^{-2} \cdot \mathrm{HD}\right)$ time in CONGEST. This yields the time complexities from Theorem 12 if $W \in o(n^c)$.

Planar Graphs The the construction for planar graphs follows the same basic scheme. However, we use a different oracle to construct the separators. By Theorem 7, we can construct 4-path separator in both HYBRID and CONGEST in arbitrary decompostions of a planar graph G. Thus, using this separator algorithm as oracle $\mathcal{O}_{\kappa}^{\mathcal{D},\epsilon}$ in the construction gives us label and tables of size:

$$\underbrace{4}_{\substack{\textit{Paths in Separator}\\(\textit{Theorem } 6)}} \cdot \underbrace{O\left(\log(nW)\right)}_{\textit{Distance Scales}} \cdot \underbrace{O\left((\epsilon^{-1}\log^4(n)\right)}_{\substack{\textit{Trees per Distance Scale}\\(\textit{Lemma } 11.8)}} \cdot \underbrace{O\left((\log^2(nW))\right)}_{\substack{\textit{Bits per Tree Label}\\(\textit{Lemma } 11.1)}} = O\left(\epsilon^{-1}\log^5(nW)\right)$$

Recall that the construction of the separators takes — no matter how we decompose the graph — $\tilde{O}(1)$ time in HYBRID and $\tilde{O}(\mathrm{HD})$ time in CONGEST. We need to apply this construction a total of $O(\log^3 n)$ times, $(\log^2 n)$ times for each application of Lemma 11.8 in Lemma 11.9. Thus, we spend a total of $\tilde{O}(1)$ time in HYBRID and $\tilde{O}(\mathrm{HD})$ time in CONGEST constructing the separators. The remaining operations are $O(\epsilon^{-1})$

approximate SetSSP computations and minor aggregations. Thus, the total complexity is $\tilde{O}(\epsilon^{-3})$ in HYBRID and $\tilde{O}(\epsilon^{-3} \cdot \mathrm{HD})$ in CONGEST, w.h.p. This proves the last statement of Theorem 12.

This section is structured as follows. We present a formal description of the algorithm behind Lemma 11.8 in Section 11.4.1. Then, in Subsubsection 11.4.2, we analyze the algorithm and prove Lemma 11.8. We conclude by showing that any additive tree cover can be turned into a full tree cover in Section 11.4.3 where we prove Lemma 11.9.

11.4.1 Constructing Additive Tree Covers

Recall that the tree cover is parameterized with a distance bound $\mathcal{D} \in [1, nW]$ and a parameter $\epsilon > 0$ that trades the number of trees with the multiplicative and additive distortion. For our algorithm, we need some *helper variables* that are based on these parameters. Note that the values for the variables are chosen with hindsight such that they can be used more easily in the analysis. We do not claim that our specific choices are optimal; they are chosen for convenience. They can likely be optimized within constant and probably even logarithmic factors. First, we define the *relaxed* distance parameter

$$\mathcal{D}' := 6400 \cdot \mathcal{D} \cdot \log^2(n)$$

and the error parameter

$$\epsilon_{pd} := \frac{1}{16000 \log^2(n)}.$$

These will be input parameters for our pseudo-padded decomposition algorithm promised by Theorem 9. This choice will ensure that a path of length $\mathcal{D}$ is not cut by the padded decomposition algorithm with probability $1 - O\left(\frac{1}{\log n}\right)$, so we can apply it $O(\log n)$ times. Further, we need the following three parameters:

$$\begin{aligned} \epsilon_p &= \epsilon/12 \\ \epsilon_t &= \epsilon/12 \\ \epsilon_s &= \epsilon_p/6400 \log^2(n) \end{aligned}$$

All these parameters will be used in different subroutines.

We can now describe the main loop of the algorithm. On a high level, our algorithm is a classical recursive divide-and-conquer algorithm that creates tree covers for subgraphs of decreasing size. For the *divide* step, we use a padded decomposition to create subgraphs of diameter $\mathcal{D}' \in O(\mathcal{D} \log^2 n)$. The *conquer* works in five synchronized phases that compute and remove a weak $(\mathcal{D}', \epsilon_s)$-separator, which ensures that the size of each subgraph shrinks by a constant factor each step. In the following, we call a node, which was not yet part of a separator, an *uncharted* node. A single recursive *conquer* step works as follows:

(Step 1) Create Partitions: Let $C_1, C_2, \ldots$ be the connected subgraphs of uncharted nodes of arbitrary diameter (where initially, it holds $C_1 := G$). Compute a pseudo-padded decomposition with diameter $\mathcal{D}' := 6400 \cdot \mathcal{D} \cdot \log^2 n$ and error parameter ϵ_{pd} in each subgraph using the algorithm from Theorem 9. We choose *all* uncharted nodes as possible centers of clusters $\mathcal{X}$. Thus, each node is covered by at least one (namely itself) at most n nodes. The resulting partitions are connected subgraphs $P_1, P_2, \ldots$ with diameter at most $\mathcal{D}'$.

(Step 2) Create Weak Separators in All Partitions: In each partition P_i, we compute a separator. As the distance parameter for the separator, we choose $\mathcal{D}'$, and for the approximation parameter, we pick ϵ_s. This results in a weak κ-path $(\mathcal{D}', \epsilon_s)$-separator that consists of κ paths of length at most $4\mathcal{D}'$ and nodes in distance at most $\epsilon_s \mathcal{D}' = \epsilon_p \mathcal{D} = O(\epsilon \mathcal{D})$ to these paths. Note that these paths (except the first) are not necessarily (approximate) shortest paths within partition P_i.

(Step 3) Create Portals on Separators: On each separator path computed in the previous step, create a collection of portals with distance $\epsilon_p \cdot \mathcal{D}$ to each other. As the length of each path is bounded by $4\mathcal{D}' = 4 \cdot (6400 \cdot \mathcal{D} \cdot \log^2 n) = 25600 \cdot \mathcal{D} \cdot \log^2 n$, there are $25600 \cdot \epsilon_p^{-1} \cdot \log^2 n$ portals per path. This sums up to $O(\kappa \cdot \epsilon_p^{-1} \cdot \log^2 n)$ portals. The portals can be efficiently computed using the algorithm from Lemma 10.25.

(Step 4) Grow Trees from Portals: In this step, we compute the actual trees of the tree cover by performing a $(1 + \epsilon_t)$ approximate SetSSP from each portal within their respective partition. Given our bound on the portals, there are $O(\kappa \cdot \epsilon_p^{-1} \cdot \log^2 n)$ approximate SetSSP computations.

(Step 5) Prepare Next Recursion: Each uncharted node on the separator removes itself and its incident edges from the graph. These nodes will not partake in future iterations. All remaining uncharted nodes compute their respective connected component for the next recursion.

As we removed weak $\mathcal{D}'$-separators $S_1, S_2, \ldots$ in each connected subgraph $C_1, C_2, \ldots$, each node has at most $(7/8) \cdot n$ other nodes in distance $\mathcal{D}'$ in the resulting partitions $C_1 \setminus S_1, C_2 \setminus S_2, \ldots$, Therefore, the invariant required in the first phase of the next stop holds again. We repeat this process until all uncharted subgraphs are empty. As we remove a separator in each step, the size of the uncharted components shrinks by a factor of $7/8$ each round. Thus, the process can be stopped after $8 \log n$ recursions as

$$n\,(1 - 1/8)^{8 \log n} \leq n(1/e)^{\log n} = n/n = 1.$$

11.4.2 Proof of Lemma 11.8

One can easily verify that each step can be executed in $\tilde{O}(\kappa \cdot \epsilon^{-1})$ rounds of minor aggregations and approximate SetSSP computations because the algorithm mostly relies on subroutines we have analyzed before. Formally, it holds:

Lemma 11.10. *Given an algorithm $\mathcal{O}_\kappa^{\mathcal{D}',\epsilon_s}$ for a weak κ-path $(\mathcal{D}', \epsilon_s)$-separator, each recursive step of the algorithm can be implemented with one execution of $\mathcal{O}_\kappa^{\mathcal{D}',\epsilon_s}$, further $O(\kappa \cdot \epsilon^{-1} \cdot \log^2 n)$ minor aggregations, and $O(\kappa \cdot \epsilon^{-1} \cdot \log^2 n)$ $(1 + O(\epsilon/\log^3))$-approximate* SetSSP *computations.*

Proof. In Step 1 we create a pseudo-padded decomposition using the algorithm promised by Theorem 9. As we parameterize the algorithm with an error parameter of $1/6400 \log^2 n$, it can be implemented with $\tilde{O}(1)$ $(1 + 1/6400 \log^2 n)$-approximate SetSSP computations. In Step 2, we construct the separator using $\mathcal{O}_\kappa^{\mathcal{D}',\epsilon_s}$ and do nothing else. In Step 3, we construct portals on all separator paths. For a single path (per connected component) this can be done in $\tilde{O}(1)$ minor aggregations using the algorithm from Lemma 10.25. As we have κ paths (per connected component) on which we compute the portals sequentially, the total complexity rises to $\tilde{O}(\kappa)$. In Step 4, we compute a $(1 + \epsilon_t)$-approximate shortest path from each portal. This step consumes the lion's share of the runtime. Again, we can compute one approximate shortest path per component in parallel. As we have $O(\epsilon^{-1} \log^2 n)$ portals per path and κ paths per connected component, we require a total of $O(\kappa \cdot \epsilon^{-1} \cdot \log^2 n)$ SetSSP computations. The final step is purely local and, thus, the lemma follows. □

This proves the postulated runtimes. Therefore, it remains to prove that the resulting tree cover has the promised properties, i.e., the approximation and overlap guarantees. First, we show that there must be a tree with low distortion for each pair of nodes. It holds:

Lemma 11.11. *Let $v, w \in V$ be a pair of nodes with distance $d_G(v, w) = \mathcal{D}$. Then, **with constant probability**, the tree growing process with parameter $\mathcal{D}$ creates a tree T with root $r \in V$, such that*

$$\tilde{d}_T(v, w) := d_T(v, r) + d_T(r, w) \leq (1 + \epsilon) \cdot d_G(v, w) + \epsilon\mathcal{D}.$$

Proof. Consider a shortest path $P_{vw} := (v, \ldots, w)$ between v and w. For this proof, we pessimistically assume that there is only one such path, although there could be several. We say that the path is intact (in step i) iff all nodes of P_{vw} are contained in the same connected component (in step i). Otherwise, the path is split. In each recursive step, two events can cause the path to be split. Either the padded decomposition places nodes of P_{vw} in different partitions *or* one or more nodes of P_{vw} lie on the separator computed in this step. We call the former a *bad* split and the latter a *good* split. Our proof consists of two parts: First, we will show that the probability of a bad split is very low. Then, we argue that — under the condition that no bad split occurs — the procedure *must* create a tree with the desired properties, w.h.p.

We begin with a probability of a bad split in a fixed step i. Let C be the connected component containing P_{vw} in step i and let $P(v)$ be the partition containing v. Note that all nodes of P_{vw} are contained in the ball $B_C(v, \mathcal{D})$ as P_{vw} is intact per definition. The path stays intact if this ball is also in $P(v)$. Thus, we compute the probability that the complete ball is contained in the same partition as v using Theorem 9. For this, we need to determine the parameter γ, i.e., the ratio between the partition's diameter and $\mathcal{D}$. Recall that we execute the padded decomposition with distance parameter $\mathcal{D}' := 6400\mathcal{D} \log^2(n)$. Therefore, we have $\mathcal{D} := 1/6400 \log^2(n) \mathcal{D}'$. In particular, it holds that $\gamma := 1/6400 \log^2(n) \in o(1)$, so it is below the upper bound required by Theorem 9. Further, we pick all nodes as potential cluster centers, so we have $\tau \leq n$. Using error parameter $\epsilon_{pd} = {}^{1}/_{16000 \log n}$, it holds by Theorem 9 that:

$$
\begin{aligned}
\textbf{Pr}[B_G(v, \gamma \mathcal{D}') \subset P(v)] &\geq e^{-16 \log(n)(\gamma + 5\epsilon_{pd})} - 160\epsilon_{pd} \log n \\
\text{Using } \gamma = {}^{1}/_{6400 \log^2 n}: & \\
&\geq e^{-16 \log(n)\left(\frac{1}{6400 \log^2 n} + 5\epsilon_{pd}\right)} - 160\epsilon_{pd} \log n \\
\text{Using } \epsilon_{pd} = {}^{1}/_{16000 \log^2 n}: & \\
&\geq e^{-16 \log(n)\left(\frac{1}{6400 \log^2 n} + 5 \cdot \frac{1}{16000 \log^2 n}\right)} - \frac{160 \log n}{16000 \log^2 n} \\
&\geq e^{-\log(n)\left(\frac{1}{400 \log^2 n} + \frac{1}{200 \log^2 n}\right)} - \frac{160 \log n}{16000 \log^2 n} \\
&\geq e^{-{}^{2}/_{100 \log(n)}} - {}^{1}/_{100 \log(n)} \\
\text{Using } e^x \geq 1 + x: & \\
&\geq 1 - {}^{3}/_{100 \log(n)} \geq 1 - {}^{1}/_{16 \log(n)}
\end{aligned}
$$

Finally, a simple union bound over all $8 \log n$ recursive steps yields a constant upper bound for the probability of a bad split, namely:

$$
\begin{aligned}
\textbf{Pr}[\textbf{Bad Split}] &\leq \textbf{Pr}\left[\bigcup_{i=1}^{8 \log n} \{B_G(v, \gamma \mathcal{D}') \not\subset P_i(v)\}\right] \\
&\leq \sum_{i=1}^{8 \log n} \textbf{Pr}[B_G(v, \gamma \mathcal{D}') \not\subset P_i(v)] \\
&\leq \sum_{i=1}^{8 \log n} (1 - \textbf{Pr}[B_G(v, \gamma \mathcal{D}') \not\subset P_i(v)]) \\
&= \sum_{i=1}^{8 \log n} 1 - \left(1 - \frac{1}{16 \log n}\right) = \frac{8 \log n}{16 \log n} = \frac{1}{2}
\end{aligned}
$$

From now on, we assume that there is no bad split and continue with the second part. It remains to be argued why there must be a tree with additive stretch for each pair of nodes if there is no bad split. Recall that after $O(\log n)$ recursions, the graph is empty, w.h.p. Thus, every path is eventually split. Without bad splits, one can easily verify that on some level of the recursion, there *must* be a good split. Otherwise, the path wouldn't have been split, and there would be a component with at least two nodes connected by an edge. This is a contradiction as we assume that subgraphs are empty. Thus, eventually, there must be a good split that adds one node of path P_{vw} to a separator.

Let u be the **first** node on the path P_{vw} that is part of some separator S. Recall that S consists of κ paths $\mathcal{P}_1, \ldots, \mathcal{P}_\kappa$ and (depending on the concrete construction) some sets $B_1, \ldots, B_\kappa$ in distance at most $\epsilon_s \mathcal{D}'$ to these paths. That means u is in distance at most $\epsilon_s \mathcal{D}' = \epsilon_p \mathcal{D}$ to some path $\mathcal{P}_l$. Now denote u' as the closest node to u on path $\mathcal{P}_l$ and u'' as the closest portal to u' and consider the distance from v to u''. We will now compute that the distance between v and w in tree $T_{u''}$ via the root u''. By the triangle inequality, we have:

$$\begin{aligned}
d_C(v, u'') &\leq d_C(v, u) + d_C(u, u'') \\
&\leq d_C(v, u) + d_C(u, u') + d_C(u', u'') \\
&\leq d_C(v, u) + d_C(u, u') + d_{\mathcal{P}_l}(u', u'') \\
&\leq d_C(v, u) + \epsilon_s \mathcal{D}' + \epsilon_p \mathcal{D} \\
&\leq d_C(v, u) + \epsilon_p \mathcal{D} + \epsilon_p \mathcal{D} \\
&\leq d_C(v, u) + 2\epsilon_p \mathcal{D}
\end{aligned}$$

Now, recall the fact that the shortest path to w in C is equal to the shortest path in G. Per definition, no node/edge of this path was removed before. Therefore, we have:

$$d_C(v, u'') = d_G(v, u) + 2\epsilon_p \mathcal{D} \tag{11.3}$$

And completely analogously, it holds for the other node w:

$$d_C(w, u'') \leq d_G(w, u) + 2\epsilon_p \mathcal{D} \tag{11.4}$$

Therefore the distance in the approximate shortest path tree rooted in u'' is:

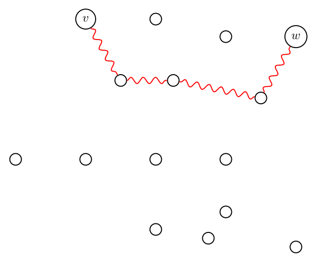
$$\begin{aligned}
\tilde{d}_T(v, w) &:= d_T(v, u'') + d_T(u'', w) \\
&\leq (1 + \epsilon_t) \left(d_C(v, u'') + d_C(u'', w) \right) \\
&\leq (1 + \epsilon_t) \left(d_G(v, u) + d_G(u, w) + 4\epsilon_p \mathcal{D} \right) && \triangleright \textit{By inequalities } (11.3) \textit{ and } (11.4) \\
&= (1 + \epsilon_t) \left(d_G(v, w) + 4\epsilon_p \mathcal{D} \right) && \triangleright \textit{As } u \in P_{vw}
\end{aligned}$$

Using our bounds for $\epsilon_p = \epsilon_t = \epsilon/12$, we conclude:

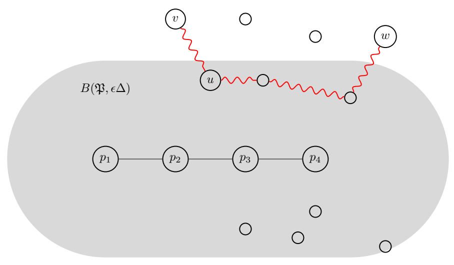
$$\begin{aligned}
\tilde{d}_T(v, w) &\leq (1 + \epsilon_t) d(v, w) + 4\epsilon_p \mathcal{D} + 4\epsilon_p \epsilon_t \mathcal{D} \\
&\leq (1 + \epsilon/12) d(v, w) + (\epsilon/3 + 4\epsilon^2/12^2) \mathcal{D} \\
&\leq (1 + \epsilon) d(v, w) + \epsilon \mathcal{D}
\end{aligned}$$

Thus, the tree rooted in u has $(1 + \epsilon)$-multiplicative and $\epsilon\mathcal{D}$-additive stretch for v and w. This was to be shown. □

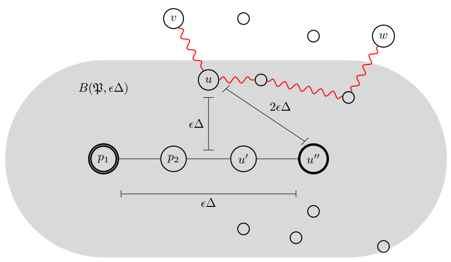
Thus, if we independently construct $O(\log n)$ such tree covers with parameter ϵ and $\mathcal{D}$, at least one of them contains a tree with the desired properties for some fixed pair $v, w \in V$, w.h.p. A union bound over all $O(n^2)$ pairs shows they all get covered, w.h.p. Therefore, it remains to bound the number of forests. Due to the divide-and-conquer nature of the algorithm, this is straighforward. It holds:



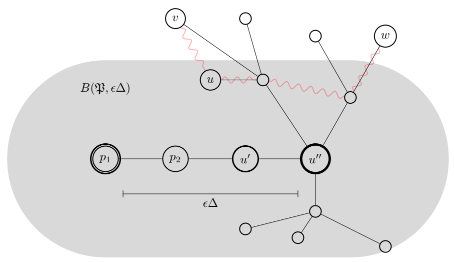
(a) We consider connected subgraphs of diameter $\mathcal{D}' \in O(\mathcal{D}\log^2 n)$. They are created with the algorithm from Theorem 9. Nodes v and w are part of the same subgraph.



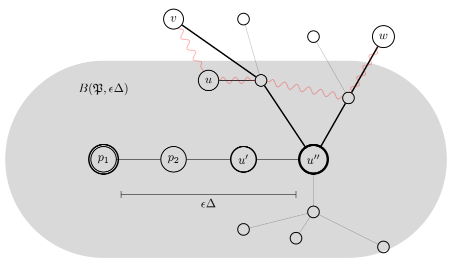
(b) We construct a weak κ-path $(\mathcal{D}', \epsilon_s)$-separator. Node u is on the shortest path between v and w and part of the separator.



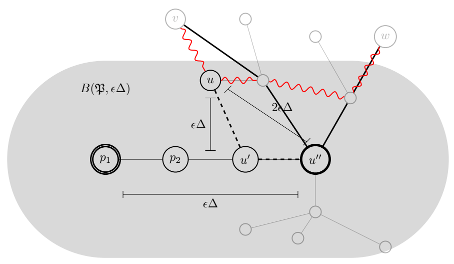
(c) We mark portal nodes in distance $\epsilon_p\mathcal{D}$ on the separator paths. Here, u'' is the portal closest to u. The path between u and u'' has length $\leq 2\epsilon\mathcal{D}$.



(d) We sequentially construct a $(1+\epsilon_t)$-approximate shortest path tree from each of the $\tilde{O}(\kappa)$ portal nodes. These trees will added to the tree cover $\mathfrak{T}$.



(e) For each $v \longrightarrow w$-path that has a node u in the separator, there is an approximate shortest path tree that contains both v and w and approximates the path.



(f) The distortion of the path (via u'') in the tree depends on $(1+\epsilon_t)$, the multiplicative error of the SetSSP, and the distance between u and u'', which is only additive.

Figure 11.1: A visualization of the main ideas of our construction of the tree cover. Note that in a traditional path separator, the node u would lie directly on the separator path, which saves an addive $\epsilon_s\mathcal{D}$.

Lemma 11.12. *A tree cover $\mathcal{T}(\epsilon, \mathcal{D})$ computed by the algorithm can be divided into $O(\kappa \cdot \epsilon^{-1} \cdot \log^4 n)$ forests.*

Proof. The forests are constructed as follows. Recall that in each recursive step, we construct at most $K \in O(\kappa \cdot \epsilon^{-1} \cdot \log^2 n)$ trees in each connected component. We enumerate all trees in all connected components, i.e., each tree gets assigned a number in $[1, K]$. Using this numbering we define the series of forests $\mathcal{F} := F_{(1,1)}, F_{(1,2)}, \dots$ where forest $F_{(i,j)}$ contains all tree that get assigned number $j \in [1, K]$ in recursion $i \in [1, O(\log n)]$. Obviously, the trees in each forest $F_{(i,j)}$ are disjoint as they are picked from different connected components of their respective graph. Further, as there are at most $O(\log n)$ recursive steps and each connected component in each recursive step has K forests, there are $O(K \log n)$ forests in total. Finally, as we indepedently repeat the algorithm $O(\log n)$ times to compute $O(\log n)$ covers, the final number of forest is $O(K \log^2 n)$. As $K \in O(\kappa \cdot \epsilon^{-1} \cdot \log^2 n)$, the lemma follows. □

11.4.3 From Additive Tree Covers To Hierarchical Tree Covers

The algorithm for constructing a *full* tree cover is very straightforward. We use the additive tree cover algorithm to compute several fixed distances and combine them into a full cover. To be precise, we simply double the diameter of each new tree cover until we reach the maximum possible diameter. In particular, we choose $d^* := \log(nW)$, i.e., d^* is the logarithm of the biggest possible path length. Recall that $W \in O(n^c)$ is the largest possible weight of an edge and there can be at most n edges to a simple path. The error parameter for all coverings $\mathcal{T}_1, \dots, \mathcal{T}_{d^*}$ is $\epsilon/3$ where ϵ is our goal approximation. The whole procedure is summarized in Algorithm 3.

Algorithm 3 HierachialCover($G := (V, E, \ell), \epsilon$)

```
$\mathfrak{T} \longleftarrow \emptyset$                                  ▷ Initialize output set $\mathfrak{T}$.
$W \longleftarrow \max_{w \in E}\{w_e\}$                                  ▷ Maximum edge weight in $G$.
$d^* \longleftarrow \lceil \log nW \rceil$                                  ▷ Note $d^* \in O(\log n)$ as $W \in o(n^c)$
for $i \leftarrow 0, \dots, d^*$ do
    $\mathcal{D}_i \longleftarrow 2^i$
    $\mathcal{T}_i \longleftarrow$ TreeCover($G, \mathcal{D}_i, \epsilon/6$)          ▷ From Section 11.4 (Lemma 11.7)
    $\mathfrak{T} \longleftarrow \mathfrak{T} \cup \{\mathcal{T}_i\}$
return $\mathfrak{T}$
```

We will now prove that this simple construction fulfills the properties postulated in the definition. In particular, we must show that for each pair $v, w \in V$ there is a tree that approximates the distance between *and* there are at most $\tilde{O}(\kappa \cdot \epsilon^{-1})$ forests in the cover. We prove the two properties separately:

1. Let $\mathfrak{T} := \mathcal{T}_1, \dots, \mathcal{T}_{d^*}$ be a series of tree covers as defined above. Then, for each pair of nodes $v, w \in V$ with distance $d(v, w)$ there is a tree T with distortion $d_T(v, w) \leq (1 + \epsilon)d(v, w)$. To see this, first note that $d(v, w) < nW$ by definition of W and therefore, it holds $\log(d(v, w)) < \log(nW)$. By construction of $\mathfrak{T}$, there must be a tree cover $\mathcal{T}_i$ with $i := \lceil \log(d(v, w)) \rceil$ as $\log(d(v, w)) < \log(nW)$. This tree cover has diameter $\mathcal{D}_i \in [d(v, w), 2d(v, w)]$. Using the definition of our tree cover, we conclude that there must be a tree $T \in \mathcal{T}_i$ for which it holds:

$$d_T(v, w) \leq (1 + \epsilon/3)d_G(v, w) + \epsilon/3\mathcal{D}_i$$

Now, we use the fact that $\mathcal{D}_i < 2d(v,w)$ and see:

$$\begin{aligned} d_T(v,w) &\leq d(v,w) + {}^{\epsilon}/_{3}d_G(v,w) + {}^{2\epsilon}/_{3}d_G(v,w) \\ &= d(v,w) + ({}^{\epsilon}/_{3} + {}^{2\epsilon}/_{3}) \cdot d_G(v,w) \\ &= d(v,w) + \epsilon \cdot d_G(v,w) \\ &= (1+\epsilon)d(v,w) \end{aligned}$$

Thus, this tree provides us with a routing path of the desired stretch.

2. The number of forests follows directly from the fact that d^*, the total number of all tree covers, is logarithmic. Given that each tree cover contributes $\tilde{O}(\kappa \cdot \epsilon^{-1})$ trees for each node, w.h.p., it easy to see that in $\mathfrak{T}$, each node is in at most $\tilde{O}(\kappa \cdot \epsilon^{-1} \cdot d^*)$ trees. As $d^* \in O(\log n)$ for $W \in O(n^c)$, the total number is trees is still $\tilde{O}(\kappa \cdot \epsilon^{-1})$.

Let us end the section with the time complexity of this whole construction. Since we need to repeat the algorithm from Lemma 11.8 for $O(\log(n \cdot W))$ distance values, the total time complexity is also $O(\log(n \cdot W))$ times the complexity of said algorithm.

11.5 Related Work

In recent years, there have been several breakthroughs in the *distributed* computation of such routing schemes in almost optimal time. The concrete time bounds depend on the model of computation. In this section, we compile the most recent and influential results from the sequential model (Section 11.5.1), the CONGEST model (Section 11.5.2), and the HYBRID model (Section 11.5.3).

11.5.1 Compact Routing Schemes in the Sequential Model

There is a plethora of compact routing schemes that can be computed in polynomial time in the classical sequential model. Peleg and Upfal were the first to consider the trade-off between space and stretch [PU89] showing that any scheme with stretch k requires $\Omega(n^{1/k})$ memory for the routing tables. They assume that Erdös' girth conjecture is true and there are graphs with $O(n^{1+1/k})$ and girth k. Shortly after, Awerbuch et al. devised a scheme with stretch $O(9^k)$ and $\Omega(n^{1/k})$ memory per node. This was later improved to $4k-5$ by Throrup and Zwick in a highly influential publication [TZ01]. To the best of our knowledge, the current state of the art was developed by Chechik [Che13]. Her work presents a scheme with stretch $3.85k$ and, thus, almost achieves the fundamental lower bound of stretch $2k-1$ with memory $O(n^{1/k})$. Besides these results for general graphs, there are also specialized schemes designed for numerous families of network topologies. These schemes circumvent the impending lower bounds by Peleg and Upfal as they exclude the pathological graphs used in their construction. These schemes achieve (almost) arbitrary stretch with a mere fraction of the memory required for general graphs and, therefore, offer significant improvements on the stretch-space tradeoff over universal routing schemes. The families in question include trees [TZ01], where one can devise schemes with stretch 1 (as there is only one path between two nodes in a tree) and logarithmic memory, as well as planar graphs [Tho04] and fixed-minor-free graphs in general [AG06] where one has to stretch $1 + \epsilon$ and labels and tables of size $O(\epsilon^{-1})$. For the minor-free graphs, the hidden constants depend on the size of the minor. Typically, this is assumed to be a (small) constant independent of n. Furthermore, schemes for graph classes can be used to model wireless networks. In this category, we have schemes for graphs with low doubling dimension (ddim) [AGGM06] and (arguably) the most famous graph family that model wireless networks, namely Unit Disc Graphs (UDG) [MW20]. The ddim of graph G is defined as follows: Suppose that G has a ddim of α, then it holds for any node $v \in V$ and any parameter $r > 0$ that the ball $B(v, 2r)$ contains at most $\alpha \cdot B(v, r)$ nodes. An often used example of a metric with low doubling dimension is the Euclidean plane $\mathbb{R}^2$, which has ddim of 8. In a UDG on the other hand, all nodes are modeled points in $\mathbb{R}^2$, but are only connected (i.e., have an edge) to all nodes in distance at most 1. In both cases there are routing schemes with stretch $1+\epsilon$ and polylogarithmic label and table sizes. Finally, there are also compact routing schemes for power-law graphs[CSTW12]. Power-law graphs constitute an important family of networks appearing in various real-world scenarios such as the Internet. In a power-law graph, the number of nodes with degree x is proportional to $x^{-\beta}$, for some constant β. The power-law exponent for many real-world networks is in the range between $\beta \in (2, 3)$. Here one can find a routing scheme with stretch 3.

11.5.2 Compact Routing Schemes in the CONGEST Model

Let us now consider related results for CONGEST. We already mentioned several of them in the beginning, but for the sake of completeness, we mention them here as well. As mentioned, for general graphs, the best we can hope for is a stretch of $2k - 1$ with routing tables of size $\tilde{O}(n^{1/k})$ due to Erdös' girth conjecture [TZ01]. The good news is that this was nearly matched in a series of algorithmic results [EN18b, EN16, LP13, LP15, LPP19]. In particular, [EN18b] gives a solution with stretch $O(k)$, routing tables of size $\widetilde{O}(n^{1/k})$, routing labels of size $\widetilde{O}(k)$ that can be computed in $\widetilde{O}\big(\text{HD} + n^{1/2+1/k}\big) \cdot n^{o(1)}$ rounds. For more restricted graphs, Izumi, Kitamura, Naruse, and Schwartzman [IKNS22] introduce a fully polynomial-time distributed tree decomposition algorithm. It yields a decomposition of width $O(\tau^2 \log n)$ in $\tilde{O}(\tau^2\text{HD} + \tau^3)$ rounds, where τ is the graph's treewidth. Additionally, they present a novel concept of a stateful walk constraint, which naturally defines a set of feasible walks in the input graph based on their local properties. Thus, for CONGEST, they almost match all relevant lower bounds. Further, for planar graphs, Li and Parter present an exact routing scheme that can be computed in $\tilde{O}(\text{HD}^2)$ time [LP19] with labels and tables of size $\tilde{O}(\text{HD})$. Finally, Table 11.1 presents an overview of all relevant work in CONGEST.

11.5.3 Compact Routing Schemes in the HYBRID Model

We conclude with some related work in the HYBRID model. We begin with some lower bounds: In a recent article, Kuhn and Schneider [KS22] prove that it takes $\tilde{O}(n^{1/3})$ rounds to compute exact routing schemes with labels of size $O(n^{2/3})$ on unweighted graphs in the HYBRID model and provide an algorithm that matches this. They also give **polynomial** time lower bounds of $\Omega(n^{1/f(k)})$ for routing schemes with stretch k on weighted graphs. Here, $f(k)$ is a function polynomial in k. They provide algorithms with near-matching upper bounds for this setting as well. As in the lower bound for the memory requirement, the lower bound graph is a dense graph of high girth. Again, the lower bound can be circumvented by considering special families of graphs.

Ref.	Runtime	Stretch	Table size	Label size	Comment
[GZ22a]	$\widetilde{O}\left(\mathrm{HD}+n^{1/2}\right)$	$O(1)$	Any	Any	Lower Bound
[LP13]	$\widetilde{O}\left(\mathrm{HD}+n^{1/2+1/4k}\right)$	$6k-1+o(1)$	$\widetilde{O}\left(n^{1/2+1/k}\right)$	$O(k\log n)$	General Graphs
[LP15]	$\widetilde{O}\left(\mathrm{HD}+n^{1/2+1/k}\right)$	$4k-3$	$\widetilde{O}\left(n^{1/k}\right)$	$O(k\log n)$	General Graphs
[LPP19]	$\widetilde{O}\left(\left(\mathrm{HD}+n^{1/2+1/k}\right)\cdot\alpha\right)$	$4k-3+o(1)$	$\widetilde{O}\left(n^{1/k}\right)$	$O(k\log^2 n)$	Extension of [LP13]
[EN16]	$\widetilde{O}\left(\left(\mathrm{HD}+n^{1/2+1/k}\right)\cdot\beta\right)$	$4k-5+o(1)$	$\widetilde{O}\left(n^{1/k}\right)$	$O(k\log^2 n)$	Optimal Runtime
[EN18b]	$\widetilde{O}\left(\left(\mathrm{HD}+n^{1/2+1/k}\right)\cdot\gamma\right)$	$4k-5+o(1)$	$\widetilde{O}\left(n^{1/k}\right)$	$O(k\log n)$	Optimal Memory
[LP19]	$\widetilde{O}\left(\mathrm{HD}^2\right)$	1	$\widetilde{O}\left(\mathrm{HD}\right)$	$O(\mathrm{HD})$	Planar Graphs
[IKNS22]	$\tilde{O}(\tau^2\mathrm{HD}+\tau^3)$	1	$\tilde{O}(\tau^2\mathrm{HD})$	$O(\tau^2\log n)$	Treewidth τ
Thm. 12 + Lem. 1.2	$\tilde{O}\left(\epsilon^{-3}\cdot\mathrm{HD}\right)$	$1+\epsilon$	$\tilde{O}\left(\epsilon^{-1}\right)$	$\tilde{O}(\epsilon^{-1})$	Planar Graphs
Thm. 12+ Lem. 1.1	$\tilde{O}\left(\epsilon^{-3}\cdot\tau\cdot\mathrm{HD}\right)$	$1+\epsilon$	$\tilde{O}\left(\tau\epsilon^{-2}\right)$	$\tilde{O}(\tau\epsilon^{-2})$	Treewidth τ
Thm. 12+ Lem. 1.3	$\tilde{O}\left(\epsilon^{-3}\cdot g\cdot\mathrm{HD}\right)$	$1+\epsilon$	$\tilde{O}\left(g\epsilon^{-2}\right)$	$\tilde{O}(g\epsilon^{-2})$	Euler-genus g
Thm. 12+ Lem. 7.8	$\tilde{O}\left(\epsilon^{-3}\cdot r\cdot\mathrm{HD}\right)$	$1+\epsilon$	$\tilde{O}\left(f(r)\epsilon^{-2}\right)$	$\tilde{O}(f(r)\epsilon^{-2})$	K_r-free
Thm. 12	$\tilde{O}\left(\epsilon^{-3}\cdot k\cdot\mathrm{HD}\right)$	$1+\epsilon$	$\tilde{O}\left(k\epsilon^{-2}\right)$	$\tilde{O}(k\epsilon^{-2})$	k-path Separable

[a] $\alpha := 2^{O(\sqrt{\log n})}$
[b] $\beta := \min\left\{2^{O(\sqrt{\log n})}, \log^{O(k)} n\right\}$
[c] $\gamma := \log^{O(\max\{k,\log\log n\})} n$
[d] $f(r)$ depends only on r

Table 11.1: An overview of the related work in CONGEST for general graphs, planar graphs, and graphs of bounded treewidth.

The first works that considered routing on special graphs in the HYBRID network model were [JKSS18] and [CKS20]. In particular, they consider a UDG with up to h so-called radio holes. These are areas enclosed by the nodes of the UDG but do not contain any edges. For the remainder, we will refer to a UDG with up to h radio holes simply as an h-UDGs. Jung et al. [JKSS18] show that for a bounded-degree h-UDG with a convex outer boundary, one can compute an abstraction of UDG in $O(\log^2 n)$ time so that paths of constant stretch between all source-destination pairs can be found. On the downside, the algorithm requires some nodes to potentially store a linear fraction of the nodes' coordinates. In [CKS20], the same authors extend on this and show that in h-UDGs where the bounding boxes[2] of the radio holes do not overlap, the same can be done with $O(h)$ memory (as long as source and target do not lie in a bounding box). However, both algorithms assume unbounded global communication, and furthermore, a simple extension of their approach to the outer boundaries of arbitrary shapes seems unlikely. We want to remark that, with unlimited global communication one could gather the topology at each node in $O(\log n)$ time and locally compute the sequential solution for UDGs. This would beat the proposed schemes in runtime, memory, and stretch. Finally, there is a very recent line of work by Coy et al. [CCF+22, CCS+23] that also considers h-UDGs but chooses more realistic communication parameters. In particular, they limit the local communication to $O(\log n)$ bits and global communication to $O(\log^2 n)$ bits, i.e., they use the same bounds as we do in this work. For 0-UDGs without holes, i.e., arbitrary polygons, [CCF+22] presents a routing scheme of stretch 36, labels and tables of size $O(\log n)$ that can be computed in $O(\log n)$ time via a deterministic algorithm. The stretch of 36 can likely be reduced to a smaller constant through a sharper analysis[3]. In the follow-up paper [CCS+23] it was further shown that for a h-UDG, the

[2] The bounding box of a hole is the smallest rectangle that fully contains the hole.
[3] This was conveyed to us in a personal communication with one of the authors.

Ref.	**Runtime**	**Stretch**	**Table size**	**Label size**	(γ, λ)	**Comment**
[KS22]	$\widetilde{O}\left(n^{1/k}\right)$	k	$O\left(n^{1/k}\right)$	Any	$(\infty, O(\log n))$	Lower Bound
[KS22]	$O\left(n^{1/3}\right)$	1	$O\left(n^{2/3}\right)$	$O(\log n)$	$(\infty, O(\log n))$	General Graphs
[KS22]	$O\left(n^{1/k}\right)$	k	$O\left(n^{1/k}\right)$	$O(\log n)$	$(\infty, O(\log n))$	General Graphs
[CCF$^+$22]	$O\left(\log n\right)$	36	$O(\log n)$	$O(\log n)$	$\left(O(\log n), O(\log^2 n)\right)$	0-UDG[a]
[CCS$^+$23]	$O\left(h^2 \log n\right)$	36	$O(h^2 + \log n)$	$O(\log n)$	$\left(O(\log n), O(\log^2 n)\right)$	h-UDG[a]
[JKSS18]	$O\left(\log^2 n\right)$	$O(1)$	NA[b]	NA[b]	(∞, ∞)	h-UDG[a]
[CKS20]	$O\left(\log^2 n\right)$	18.55	NA[b]	NA[b]	(∞, ∞)	h-UDG[a,c]
Thm. 12	$\tilde{O}\left(n^{4/k}\right)$	$O(k^2)$	$\tilde{O}\left(n^{1/k}\right)$	$O(\log^2 n)$	$\left(O(\log n), O(\log^2 n)\right)$	General Graphs
Thm. 12 + Lem. 1.2	$\tilde{O}\left(\epsilon^{-3}\right)$	$1+\epsilon$	$\tilde{O}\left(\epsilon^{-1}\right)$	$\tilde{O}(\epsilon^{-1})$	$\left(O(\log n), O(\log^2 n)\right)$	Planar Graphs
Thm. 12+ Lem. 1.3	$\tilde{O}\left(\epsilon^{-3} \cdot g\right)$	$1+\epsilon$	$\tilde{O}\left(\tau\epsilon^{-2}\right)$	$\tilde{O}(g\epsilon^{-2})$	$\left(O(\log n), O(\log^2 n)\right)$	Euler-genus g
Thm. 12 + Lem. 1.1	$\tilde{O}\left(\epsilon^{-3} \cdot \tau\right)$	$1+\epsilon$	$\tilde{O}\left(\tau\epsilon^{-2}\right)$	$\tilde{O}(\tau\epsilon^{-2})$	$\left(O(\log n), O(\log^2 n)\right)$	Treewidth τ
Thm. 12 + Lem. 1.4	$\tilde{O}\left(\epsilon^{-3} \cdot r\right)$	$1+\epsilon$	$\tilde{O}\left(f(r)\epsilon^{-2}\right)$	$\tilde{O}(f(r)\epsilon^{-2})$	$\left(O(\log n), O(\log^2 n)\right)$	K_r-free
Thm. 12	$\tilde{O}\left(\epsilon^{-3} \cdot k\right)$	$1+\epsilon$	$\tilde{O}\left(k\epsilon^{-2}\right)$	$\tilde{O}(k\epsilon^{-2})$	$\left(O(\log n), O(\log^2 n)\right)$	k-path Separable

[a] An h-UDG is a UDG with h radio holes, i.e., areas without edges.
[b] [CKS20, JKSS18] do not present compact routing schemes but algorithms to compute routing paths for each s, t-pair in $O(1)$ time using $O(\log^2 n)$ time preprocessing. The memory requirement might be up to $O(n)$, i.e., the nodes need to store the whole graph.
[c] The bounding boxes of radio holes must not overlap.

Table 11.2: An overview of the related work in the HYBRID model. All of the routing schemes assume that the nodes are able to perform a *handshake* with the target, i.e., exchange one control message of size $O(\log n)$ before routing the actual message. This is a natural assumption in the HYBRID model and leads to smaller routing labels if compared to the schemes in Table 11.1.

runtime increases to only $O(h^2 + \log n)$ and table size to $O(h^2 \log n)$. In other words, there is a quadratic dependence on the number of holes. As in the previous paper, the stretch is 36 and the label size is $O(\log n)$. Both papers also present *exact* routing schemes for grid graphs that are of independent interest. Note that grid graphs are planar and therefore also subject to our algorithm. However, we only produce a scheme with stretch $1 + \epsilon$ where $\epsilon > 0$. Table 11.2 presents an overview of all relevant work in HYBRID and a comparison with our results.

11.6 Conclusion & Future Work

In this chapter, we presented algorithms that construct compact routing schemes in the CONGEST and the HYBRID model. Our results can be divided into two areas. First, we proved that in the HYBRID model, we can construct a compact routing scheme with constant stretch for any undirected graph (with polynomially bounded edge weights) in sublinear time, w.h.p. For a stretch of $O(\log^2 n)$, it requires only polylogarithmic time. Thereby, it massively outperforms all comparable algorithms in CONGEST model that require $\Omega(\mathrm{HD} + \sqrt{n})$ time for similar bounds. In particular, our algorithm only uses polylogarithmic communication in both the global and the local mode. In earlier works, the author either assumed unlimited local communication [KS22] or a restricted topology [CCS$^+$22, CCS$^+$23, JKSS18, CKS20]. Thus, our algorithm demonstrates the massive impact that only a tiny amount of global communication has on the computation of routing schemes. Second, we presented a compact routing scheme with a stretch $1+\epsilon$ scheme for planar graphs and universally k-path separable graphs. The latter class includes graphs of bounded treewidth, graphs of bounded Euler-genus, or graphs that exclude a fixed minor K_r. Our schemes can be constructed in almost optimal time except for some (possibly large) polylogarithmic factors in CONGEST and HYBRID. This answers the fundamental question of whether compact routing schemes with arbitrarily small stretch can be efficiently computed in a distributed fashion for a plethora of restricted graph classes.

That being said, we must remark once again that our results are purely theoretical. Although comparing favorably to other distributed algorithms, both the polynomial dependency on ϵ and the logarithmic factors in the runtime and size of the routing tables are not optimal and likely too large for actual practical usage. They could be improved in future work. However, this likely requires new techniques and ideas, particularly a more efficient separator construction. Our weak separator construction adds a multiplicative ϵ^{-1} that can be avoided when using a proper path separator (as we do for planar graphs). However, even for planar graphs, we have a linear dependency on ϵ^{-1} in the size of the routing table. The work of [KST13] shows that in a planar graph, it suffices to store $O(\log \epsilon^{-1} \cdot \log n)$ bits on average. It is unclear if their construction can be translated to the distributed setting, but it provides a starting point. As with our previous result, our constructions can also be improved without touching the actual algorithms. A faster algorithm for approximate SetSSP would immediately speed up our algorithm as well. Further, proving that K_r-free graphs are poly (r)-separable leads to faster runtimes and smaller tables for K_r-free graphs.

Finally, we believe that our algorithms have applications beyond the construction of *compact* routing schemes and may lead to better distributed algorithms for other types of routing schemes. In the remainder, we name two promising examples.

Fault-Tolerant Routing Schemes A possible direction for future work could be the distributed construction of fault-tolerant routing schemes for arbitrary and restricted graphs. Here, one is interested in a routing scheme that tolerates the failure of nodes and edges. That means, even if, say, f nodes or edges fail, the scheme still finds a competitive path between source and target. Ideally, the stretch and the size of the routing table only increase by a factor that polynomially depends on f. This topic has been heavily researched in the past couple of years, leading to a good understanding of the problem [DP21, IEWM23, PPP24, LPS24]. However, to the best of our knowledge, existing algorithms do not present efficient distributed constructions. The runtime is near-linear in the number of edges, namely $\tilde{O}(m)$. However, the hop diameter HD, which is typically much smaller than m, is conjectured to be the optimal runtime. Perhaps our techniques can help develop efficient distributed implementations.

Oblivious Routing Schemes: By design, our compact routing schemes only consider the length of the routing paths, not the congestion. This means that a-priori there could be nodes that lie on all $O(n^2)$ possible paths between nodes. As a consequence, if each node actually wanted to route a message to another node, all messages would need to cross this one specific node. Thereby, the node would experience very high congestion, which often is unfortunate in practice. Our algorithm makes no effort to mitigate this, as this is not typically considered in compact routing schemes.

In contrast to the routing schemes that we consider, so-called *oblivious* routing schemes try to minimize the congestion (cf. [CR20, GHZ21, Rac02, Rä08, Rä09] for an overview.). Before we continue, we should clarify what *minimizing the congestion* means. Note that we can easily find graphs where a congestion of $O(n)$ is unavoidable if each node wants to send a message. For example, in a balanced binary tree or a star graph. More generally, for every graph, there is an *optimal* congestion C_{OPT} of each node, which might be up to $O(n)$. A good oblivious routing scheme tries to get as close to this optimal congestion as possible. Typically, the quality is a multiplicative factor of $\tilde{O}(1)$, i.e., the schemes have a congestion of $\tilde{O}(C_{OPT})$ where C_{OPT} is the optimal congestion.

However, we want to remark that most oblivious routing schemes that optimize congestion do not consider the stretch, i.e., the paths that the messages take are very long. Thus, developing an algorithm that optimizes *both* stretch and congestion *and* can efficiently computed in CONGEST or HYBRID is a worthwhile direction for future work. In recent article [HRG22], Haeupler, Ghaffari, and Räcke presented a *hybrid* routing scheme that combines both paradigms, i.e., it computes routing paths with (relatively) low stretch of $2^{O(\sqrt{n})}$ and (relatively) low congestion of $2^{O(\sqrt{n})}$ times the optimal congestion. In CONGEST, the scheme can, w.h.p., be constructed in $O((\mathrm{HD} + \sqrt{n})2^{O(\sqrt{n})})$ time in general graphs and $O(\mathrm{HD} \cdot 2^{O(\sqrt{n})})$ time in restricted graphs. Their algorithm for oblivious and hybrid routing schemes is based on so-called (hop-constrained) *expander decompositions*. In other words, they use entirely different techniques than we use for our routing schemes. Perhaps, with an expander decomposition tailored to restricted graphs, one can also construct better oblivious or hybrid routing schemes for restricted graphs. Our efficient construction of separators can possibly also help here, but this is highly unclear and demands further research!

Beep Boop Beep.

R2D2, Star Wars: A New Hope

12 Approximating Simple Covering Problems Via Beeping Algorithms

SetCover is a well-understood problem in both the centralized and distributed setting. In this chapter, we present a new distributed algorithm for SetCover that works in a very restricted model of communication. This makes it extremly versatile and good building block for distributed optimization problems. All algorithms and techniques that are presented in this chapter have been published in the following journal article:

> Thorsten Götte, Christina Kolb, Christian Scheideler, and Julian Werthmann. Beep-and-sleep: Message and energy efficient set cover. *Theor. Comput. Sci.*, 950:113756, 2023

Given a collection of elements $\mathcal{U} := \{u_1, \dots, u_n\}$ and sets $\mathcal{S} := \{s_1, \dots, s_m\}$ with $s_i \subseteq \mathcal{U}$ the goal of SetCover is to cover all elements with as few sets as possible. A simple greedy algorithm that always picks the set that covers most elements has a logarithmic approximation factor, which is the best we can hope for a polynomial-time algorithm unless $P = NP$. SetCover has a wide variety of applications in many areas of computer science. On the one hand, it plays an essential role in the analysis of large data sets, which is often needed in fields like operations research, machine learning, information retrieval, and data mining[1]. On the other hand, it is also used in purely distributed domains like ad-hoc sensor networks. An essential task in these

[1] See, e.g., [IMR$^+$18, DIMV14, IV19, IMR$^+$17] and the references therein for a comprehensive overview on the application of SetCover.

networks is to determine a minimum set of nodes, a so-called DominatingSet, such that all nodes are within the sensor range of this set. This set can fulfill tasks like routing, collecting sensor data from neighbors, and various other jobs. Note that DominatingSet is a special case of SetCover. We can model each node as an element that the solution must cover. Further, if we add a node to the solution, it covers itself and all its neighbors. Thus, we can also model it as a set containing all these nodes. Therefore, DominatingSet is a particular case of SetCover where all nodes need to act as both sets and elements.

Recall that we are interested in *distance-based* graph problems in this part of the thesis. Thus, we will look at the following natural generalization of the DominatingSet problem.

Definition 12.1 ($\mathcal{D}$-DominatingSet)**.** *Let $G = (V, E, w)$ be a (weighted) graph and let $\mathcal{D}$ be a distance parameter. Then, an $\mathcal{D}$-DominatingSet is a subset $\mathcal{S} \subseteq V$ such that each node $v \in V$ has at least node $s \in \mathcal{S}$ in distance at most $\mathcal{D}$. Formally, for all $v \in V$, it holds:*

$$B_G(v, \mathcal{D}) \cap \mathcal{S} \neq \emptyset$$

Let $\mathcal{S}_{OPT}$ be the minimal set with these properties. Then, we say that an algorithm $\mathcal{A}$ solves $\mathcal{D}$-DominatingSet with approximation ratio α if outputs a set $\mathcal{S}_{\mathcal{A}}$ with $|\mathcal{S}_{\mathcal{A}}| \leq \alpha|\mathcal{S}_{OPT}|$.

An application of this particular flavor of the DominatingSet problem could be the following: Consider an ad-hoc sensor network where each device has an *uplink*, i.e., a separate antenna or satellite connection enables it to communicate with a controller node, a base station, etc. It can use this uplink to send its sensor data or to receive control information. However, doing this draws additional energy, which is typically considered a sparse resource in ad-hoc networks. Therefore, a reasonable trade-off seems only to let a small subset of nodes activate the uplink and ensure that these nodes are not *too far* away from any other nodes. Then, the remaining nodes can use their local low-energy communication to send and receive data to and from the uplink nodes. Note that the particular application determines whatever constitutes *too far*. In other words, we are looking for the smallest possible set of nodes that are not *too far* away from all other nodes. Thus, our definition of a $\mathcal{D}$-DominatingSet with $\mathcal{S}$ being the uplink nodes and $\mathcal{D}$ being the maximal tolerable distance to an uplink node, fits perfectly into this description.

Quite surprisingly, previous distributed algorithm for DominatingSet and/or SetCover did not specifically consider this variant of the problem. For distributed algorithms, an instance of SetCover is usually modeled as a so-called *problem graph* $G_P := \{V_{\mathcal{U}} \cup V_{\mathcal{S}}, E\}$. Each set and each element corresponds to a node in this graph. Further, for each set $s_i \in \mathcal{S}$ there is a bidirected edge $\{u_j, s_i\} \in E$ to each element $u_j \in \mathcal{U}$ it contains. These edges model bidirected communication channels between the nodes representing the set and element in the default distributed setup. Each set and element in each round of communication can send a distinct message of size $O(\log n)$ over each communication edge. Several randomized algorithms in the distributed Congest model match the optimal approximation ratio, w.h.p., and have a near-optimal runtime of $O(\log^2 \Delta)$ where Δ is the maximum degree of the problem graph [JRS02, KW03]. To the best of our knowledge, in all popular distributed solutions to SetCover, all sets/elements extensively communicate with their neighborhood and send (possibly distinct) messages to all their neighboring sets and elements. However, in our case, the **the communication graph is different from the problem graph!** While we are are looking for a

DominatingSet in $G^{(\mathcal{D})} := (V, E)$ where $E^{(\mathcal{D})} := \{(v, w) \mid w \in B_G(v, \mathcal{D})\}$, we can only communicate in G. In the LOCAL model, this does not make too much of a difference as we can use unbounded messages. In an unweighted graph, we can collect all messages from all nodes in the distance $\mathcal{D}$ within $\mathcal{D}$ rounds. We would only have a slowdown of $\mathcal{D}$ compared to the *local* solutions that only communicate with neighboring nodes. In more restricted models like CONGEST, the situation is different. Here, we cannot trivially extend existing algorithms to distance $\mathcal{D}$. To be precise, we cannot receive a *distinct* message from all nodes in the distance $\mathcal{D}$ in $\mathcal{D}$ rounds as the communication along an edge is restricted. Thus, the communication between nodes and the nodes they can potentially cover is much more restricted and we have to *compress* the information we want to send. In spite of this, we show:

Theorem 13. *$\mathcal{D}$-DominatingSet in CONGEST*

Let $\mathcal{D} \geq 1$ be a distance parameter, and let $G = (V, E)$ be an unweighted graph. Suppose that for all nodes $v \in V$, it holds $B_G(v\mathcal{D}) \leq \Delta$, then there is a randomized algorithm in the CONGEST model that solves $\mathcal{D}$-DominatingSet in $O(\mathcal{D} \cdot k^3)$ rounds with expected approximation ratio $O(\Delta^{\frac{3}{k}} \log^2 \Delta)$ where $k > 3$ is an parameter. Both k and Δ must be known to all nodes.

The algorithm sends only 1-bit messages and does not require to send distinct messages to all neighbors.

Note that the algorithm can trade runtime for approximation ratio. This trade-off makes it particularly practical for sensor networks, as their topology is often rapidly changing due to sensor nodes moving, shutting down, or rebooting. Our algorithm can quickly react to these changes by computing a new non-trivial solution in constant time. However, for instances of high degree, when the devices are placed densely, is approximation ratio may still be too high. Existing solutions for special graph classes (see, e.g.,[SRS08, YJY+15] for graphs embedded in the euclidean plane) have far better approximation ratios as they can exploit specific properties of the problem graph. On the flip side, they have at least a logarithmic runtime, which cannot trivially be reduced. Thus, our algorithm is suitable in cases where the degree is not too high, and a constant runtime is more important than an optimal approximation. Finally, note that this work focuses on proving the feasibility and not optimizing the *logarithmic* factors in message complexity and approximation ratios, i.e., we explicitly do not claim our factors to be optimal.

The remainder of this section is structured as follows. First, we present some preliminaries in Section 12.1. In particular, we introduce the so-called Beeping model, a simple model for ad-hoc networks that relies only on carrier sensing. This model is *weaker* than CONGEST, but in contrast to CONGEST, a Beeping algorithm for a nodes 1-neighborhood can easily be transformed in an algorithm for a nodes $\mathcal{D}$-neighborhood. In fact, this is exactly what we will do. Then, in Section 12.2, we present our main technical contribution, an adapted version of Jia et al.'s. DominatingSet algorithm from [JRS02] for the aforementioned Beeping model. We change this algorithm in two significant ways: First, we massively reduce the messages that need to be exchanged by replacing all instances where nodes are counted through randomized approximations. These approximations only use a fraction of the messages. Second — and more importantly — we completely replace the mechanism that lets nodes decide to join the solution. For this, we use geometrically distributed starting times where sets add themselves when their time has come, and they have a certain threshold of uncovered neighbors. This approach does not require any additional messages. Note that the independent work of [GMRV20] uses a very similar technique[2]. Still, their analysis does not provide all the properties we need for our problem. Finally, we present some related work in Section 12.3 and conclude the chapter with some ideas for future work in Section 12.4.

12.1 The Beeping Model

In wireless ad-hoc networks, one needs to take special care of the limited battery life of the nodes. In particular, sending messages should be reduced to a minimum due to the high energy requirements of the radio module. In practical ad-hoc networks, the number of messages is not the limiting factor. Instead, it is much more energy-efficient to only send a single signal, a so-called Beep, to all neighboring nodes. Further, nodes can only distinguish if at least one or none of their neighbors beeped to save energy, i.e., they only rely on carrier sensing. These considerations have been formalized in the so-called Beeping model[CK10]. In this work, we consider the following (standard) variant of the Beeping model [CK10, DBB18]: We consider a fixed communication graph $G := (V, E)$. Note that the nodes do not know their exact degree, and nodes have *no* identifiers. Time proceeds in so-called *slots*. In each slot, a node can either *beep* or *listen*. If a node *listens* and any subsets of its neighbors in G *beeps*, the *listening* node receives a Beep. It can neither distinguish which neighbors beeped nor how many neighbors beeped, i.e., it only relies on carrier sensing. Further, a node *cannot* simultaneously beep and listen but must choose one of the two options. All nodes wake up in the same slot, i.e., we consider the Beeping model with simultaneous wake-up. We believe that our algorithm can also be extended for wake-up-on-beep as each node only needs to be in sync with neighboring nodes. If nodes do not wake up in the same round, their internal counters only differ by 1 as each node wakes up one slot after its earliest neighbor. In this case, there are some standard tricks to simulate a single slot of a simultaneous wake-up algorithm within 3 slots [CK10, DBB18].

For this variant of the model, the following holds:

Lemma 12.1 (Folklore). *Let G be an unweighted graph. Then, every Beeping algorithm on* $G^{(\mathcal{D})} := (V, E)$ *where* $E^{(\mathcal{D})} := \{(v, w) \mid w \in B_G(v, \mathcal{D})\}$ *can be simulated on G in* $\mathcal{D}$ *rounds of CONGEST.*

In the simulation, all nodes can beep and listen simultaneously.

Proof. For the CONGEST part, each beep simply is relayed for $\mathcal{D}$ steps. To be precise, each slot simulated in $\mathcal{D}$ rounds in CONGEST. That is, if a node $v \in V$ beeps in given slot, it sends a 1-bit message to all its neighbors in G. Then, all nodes that received one or more such messages, send a 1-bit message to all of their neighbors. This continues for $\mathcal{D}$ rounds. Thus, a given node $v \in V$ will receive the 1-bit message if and only if a node in $B_G(v, \mathcal{D})$ *beeped*. It cannot tell, which node beeped (unless it happens to be its direct neighbor) nor how many nodes have beeped. Thus, it has the same information as in the Beeping model. Clearly, this also holds for the nodes that beeped so that they can beep and listen simultaneously. □

[2]Instead of using geometric starting times, they continuously increase a set's probability to join until it joins or does cover enough elements. From a probabilistic point of view, this is (almost) equivalent to picking geometric starting times.

12.2 An Efficient SetCover-Algorithm for the Beeping-Model

We will now describe our first algorithm, which we dub the Beep-And-Sleep algorithm, as most sets and elements will be idle during the execution. We consider a fixed communication graph $G := (V_S \cup V_{\mathcal{U}}, E)$ with $V_S = m$ and $V_{\mathcal{U}} = n$. Each set $s \in V_S$ has a bidirected edge $\{s, e\} \in E$ to each element $e \in V_{\mathcal{U}}$ it contains. Each node can only communicate with its neighbors in G. Further, all nodes know Δ, the maximum degree of G. This assumption can be replaced by a polynomial upper bound, i.e., an approximation of $\log \Delta$, which would only slow the algorithm down by a constant factor.

Given these assumptions, we show the following:

Theorem 14. *SetCover in the BeepingModel*

There is a randomized algorithm in the Beeping model that solves SetCover in time $O(k^3)$ with expected approximation ratio $O(\Delta^{\frac{3}{k}} \log^2 \Delta)$ where $k > 3$ is a parameter known to all nodes.

Thus, even for a constant k, our algorithm achieves a non-trivial approximation ratio. Note that this is close to the optimal ratio we can hope for with this runtime, as Kuhn et al. showed that any distributed algorithm with only local communication needs $O(k)$ time for an approximation ratio of $O(\Delta^{\frac{1}{k}})$ [KMW06]. Note that together with Lemma 12.1, this theorem implies Theorem 13. As the simulation allows listening and beeping simultaneously, each node can simulate a set and and element.

The core idea behind this algorithm is that — similar to the sequential greedy algorithm — all sets that cover the *most* uncovered neighbors at a given point in time add themselves to the solution. However, due to the constraints of the Beeping model, the uncovered elements cannot simply inform their neighboring sets that they are uncovered. If all elements were to send a Beep simultaneously, the listening sets would only learn that they have at least one uncovered neighbor but not how many. So, our algorithm must work around this. Further, since we are in a distributed setting and want to achieve a short runtime, we need to add sets simultaneously. Thus, we need to ensure that two (or more) sets we add concurrently do not *overlap* too much, i.e., cover the same elements. Hence, the main difficulty stems from the following two technical challenges in the Beeping model:

1. We need to estimate the number of uncovered neighbors correctly.
2. We need to avoid too many sets containing the same elements being added to the solution concurrently.

Note that the classical algorithm bu Jia et al.[JRS02] faced the same problems in the CONGEST model. In their algorithm, all sets that (approximately) cover the largest number of uncovered elements try to add themselves to the solution. Then, each of these sets locally computes the probability of joining the solution. This probability is based on how many elements are covered by the sets in their 2-neighborhood. To be precise, each element counts how many sets want to cover it and sends this value to all these sets. Thus, each set receives a value from all its uncovered elements. Then, the sets locally compute the median of these values and pick its inverse as their probability. This calculation requires each element and set to send and receive *distinct* messages of size $O(\log \Delta)$ to and from all of their neighbors (as all values are smaller than Δ and require $O(\log \Delta)$ bits to be encoded). Obviously, this is trivial to implement in the CONGEST model. However, in the Beeping model, this calculation cannot be done as easily because of the restricted communication. First, we would need at least $O(\log \Delta)$ slots to submit even a single value, as we can only transmit one bit per round. Thus, a simple simulation cannot achieve a runtime polynomial in k. Second, we cannot receive distinct values from different neighbors in the Beeping model. Therefore, the computation of the median also cannot be trivially simulated.

We will heavily use randomization and present a random mechanism to achieve the desired properties. We begin with a helpful definition that simplifies our notation. For any $i = k, \dots, 0$ we define

$$\Delta_i := \frac{\Delta}{\Delta^{\frac{i}{k}}} := \left(\Delta^{\frac{1}{k}}\right)^{k-i}.$$

Throughout this work, we will make two assumptions about the degree. First, we will assume that Δ_i is an integer for any choice of i. In other words, we assume Δ is a multiple of $\Delta^{\frac{1}{k}}$. This way, we do not need to carry rounding artifacts through our equations. Second, we assume w.l.o.g. that $\Delta > 4k$. Note that the largest meaningful value for k is $O(\log \Delta)$. For bigger values, the term $O(\log^2 \Delta)$ dominates the approximation ratio given in Theorem 14. As $\Delta \in \omega(\log \Delta)$, our assumption holds for instances of non-constant degree. Instances of degree $\Delta \in O(1)$ admit a trivial constant-factor approximation by adding all sets, so our result remains general.

12.2.1 Algorithm Description

Now we will describe the algorithm promised by the theorem: The algorithm runs in k phases, where in phase i, all sets that cover *approximately* Δ_i elements try to add themselves to the solution. At the beginning of each *phase*, the sets and elements do the following:

- Each *set* s draws a geometric random variable X_s with parameter $1 - 1/\Delta^{\frac{1}{k}}$. Values bigger than $4k$ are rounded down to $4k$, so $X_s \in [0, 4k]$ always.
- Each uncovered *element* u draws $4k$ variables $Y_u^1, \dots, Y_u^{4k}$ with $Y_u^\ell \in \{0,1\}$ and $\mathbf{Pr}\big[Y_u^\ell = 1\big] = \left(\frac{1}{\Delta_i}\right)$ independently and uniformly at random. If at least one of these variables is positive, we say that u is *active*. As we will see, this ensures that each set with Δ_i uncovered elements has roughly $4k$ active elements in expectation.

These variables stay fixed for the duration of the phase.

Obviously, not all sets with active elements in a given phase may be (simultaneously) added to the solution, as this generally would not lead to a good approximation. Many of these sets could cover the same active elements, so we must add them carefully and update the number of active elements after each addition. To this end, the phases are further subdivided into $4k + 1$ rounds, where only some sets try to add themselves. Here, the geometric distribution comes into play. Each set has precisely one round $\Phi_s \in [0, 4k]$ where it tries to enter the solution. The round is determined by $\Phi_s := 4k - X_s$. In all other rounds, the set will be idle and not try to add itself. This roughly means that in the first couple of rounds, only very few sets will try to add themselves (and only cover elements contained in many sets), while in the later rounds, almost all sets will try to add themselves. As we will see in the analysis, our particular choice of parameters ensures that in each round, only $\tilde{O}(\Delta^{\frac{1}{k}})$ sets

Algorithm 4 Code for Sets

```
procedure Sets($s, N_s, k, \Delta$)
    $S_s \longleftarrow 0$
    $A \longleftarrow \{\perp\}^{4k}$
    for $i := 0, \dots, k$ do
        Pick $X_s \sim Geo(1 - 1/\Delta^{\frac{1}{k}})$.
        $X_s \longleftarrow \min\{X_s, 4k\}$

        for $j := 0, \dots, 4k$ do
            for $\ell = 1, \dots, 4k$ do


                $A_\ell \longleftarrow$ Listen()

            if $(|\{A_\ell \neq \perp\}| \geq 3k)$ then
                if $(4k - X_s = j)$ then
                    Beep()
                    $S_s \longleftarrow 1$

            $A \longleftarrow \{\perp\}^{4k}$
```

Algorithm 5 Code for Elements

```
procedure Elements($e, N_e, k, \Delta$)
    $C_e \longleftarrow 0$

    for $i := 0, \dots, k$ do
        for $\ell = 1, \dots, 4k$ do
            $Y_e^\ell \sim B\left(1/\Delta_i\right)$
        for $j := 0, \dots, 4k$ do
            for $\ell = 1, \dots, 4k$ do
                if $(Y_e^\ell = 1)$ then
                    if $(C_e = 0)$ then
                        Beep()





            $l \longleftarrow$ Listen()
            If $l \neq \perp$, set $C_e \longleftarrow 1$
```

Figure 12.1: Pseudocode for BeepAndSleep algorithm for creating a SetCover. The pseudocode depicts the code for the elements (right) and the code for the sets (left). Sets and elements are synchronized. Whenever the sets beep, the elements listen, and vice versa.

try to cover the same element simultaneously in expectation. Note that this geometric mechanism that lets the sets decide to add themselves is the main difference from the classical algorithm by Jia et al. [JRS02].

We now focus on the order of events of a single round. A single round consists of $4k + 1$ slots. In the first $4k$ slots, the sets try to estimate the number of their active elements. During this time, the sets will only listen and count the number of beeps they hear. Each active *element* u (that has not yet been covered) beeps in slot $\ell \in [1, 4k]$ if it has drawn $Y_u^\ell = 1$ in the beginning and remains idle otherwise. Now consider a set that has at least Δ_i uncovered elements. Our choice of parameters implies that in each slot, there is (at least) one active element that sends a Beep *in expectation*. After these $4k$ steps, the sets decide to enter the solution. Each *set* s with $\Phi_s = j$ that received a Beep in at least $3k$ slots adds itself to the solution and beeps. In the final slot of the round, the elements listen for beeps, and if they hear a beep (sent by a set that adds itself to the solution), they consider themselves as covered. Once an element is covered, it will perform no further action. Note that in the last phase, it holds $\frac{1}{\Delta_k} = 1$, so all uncovered nodes become active and, more importantly, beep in all $4k$ slots. Thus, even if there is only one uncovered element, the neighboring sets will hear a Beep in $4k$ slots. As each set eventually tries to add itself, all elements will get covered.

The pseudocode for the algorithm is given in Figure 12.1. In the following, we elaborate on some of the notation used in the code. If a set or element wants to send a beep, it calls the eponymous method Beep(). If a set or element wishes to listen to Beeps, it calls the method Listen(), which either returns Beep if at least one neighboring node called Beep() or $\perp$ otherwise. We use $B(p)$ and $Geo(p)$ to refer to the Bernoulli and geometric distribution, respectively. In addition to the variables that store the aforementioned random variables, the nodes have little additional state information. Each set s has a variable $S_s \in \{0,1\}$ that is 1 if s part of the solution set and 0 otherwise. This variable is initially set to 0. Further, each set has array $A \in \{\perp, \textsc{Beep}\}^{4k}$ of size $4k$ that is used the count the beeps of neighboring elements received in one round. This array is reset after each round, so a set's total memory is only $O(k)$ bits. The elements have the variable $C_e \in \{0,1\}$, which takes value 1 if the element is covered and 0 otherwise. Together with the variables $Y_1, \ldots, Y_{4k}$ that store the random choices, the elements also need to store $O(k)$ bits.

12.2.2 Analysis (Proof of Theorem 14)

We will prove Theorem 14 and show that our algorithm fulfills the promised bounds. The runtime of $O(k^3)$ follows almost trivially. The algorithm is structured in k phases, each phase again in $4k+1$ *rounds*. Further, each round again consists of $4k+1$ slots, which brings the total runtime to $O(k^3)$. Since the runtime is deterministic, we only need to prove the expected approximation ratio.

Lemma 12.2. *The algorithm outputs a* $O\left(\Delta^{\frac{3}{k}} \log^2 \Delta\right)$*-approximate solution in expectation.*

Since the algorithm, for the most part, follows the algorithm by Jia et al.[JRS02], a generalized version of their core lemma also holds for our case. In particular, they showed that their algorithm only deviates by a constant factor from the greedy solution. They use the fact that each set can *exactly* count how many elements it covers, whereas our algorithms can only approximate it. Thus, we need to *parameterize* the analysis further to account for this.

Note that an element can only get covered in the last slot of a round. As we have $k+1$ phases and $4k+1$ rounds in a phase, there are $\tau := 4(k+1)^2 - 3$ rounds in total. For this analysis, we denote each round as a tuple (i,j) where $i \in [0,k]$ is the phase and $j \in [0,4k]$ is the round of that phase. Thus, when we write *round* (i,j), we mean the j^{th} round of the i^{th} phase. Note that i and j correspond to the loop counters in the pseudocode. We choose this notation because the phase i is very important for calculating the probabilities, so we need to carry it with us. In the following, we consider the round $(0,0) \leq (i,j) \leq (k,4k)$ in which element $u \in V_{\mathcal{U}}$ is covered. Since all elements are eventually covered (because in the last phase, all uncovered elements beep until one of their neighboring sets adds itself), this phase is well-defined. For each element covered by our algorithm, we define the random variables $\eta_{(i,j)}(u) \in [1,\Delta]$ and $\mu_{(i,j)}(u) \in [1,\Delta]$. We define $\eta_{(i,j)}(u)$ to be the ratio between the *best* set in u's neighbor and the worst set picked by the algorithm, i.e., the ratio by which the choice of our algorithm differs from the greedy solution. Further, let $\mu_{(i,j)}(u)$ be the random variable that denotes the number of candidates covering $u \in V_{\mathcal{U}}$ in the round where it is first covered.

Moreover, we define the functions $c_{min}^{(i,j)}(u) \in [\frac{1}{\Delta}, 1]$ and $c_{max}^{(i,j)}(u) \in [\frac{1}{\Delta}, 1]$ for each covered element $u \in V_{\mathcal{U}}$. Let $N_u \subset V_{\mathcal{S}}$ denote the sets that can cover u. Likewise, let N_s denote the elements contained in some set $s \in V_{\mathcal{S}}$. Suppose u is covered by sets $C_{(i,j)}(u) \subseteq N_u$, i.e., these sets add themselves simultaneously. Further, let $d_{(i,j)}(s)$ for $s \in N_u$ be the *span* of set s, i.e., the number of uncovered elements neighboring s. Based on this, we define $c_{max}^{(i,j)}(u)$ based on the set with *fewest* uncovered elements, i.e., the set that deviates the most from whatever set the greedy solution would have picked. It holds:

$$c_{max}^{(i,j)}(u) = \max_{s \in C_{(i,j)}(u)} \frac{1}{d_{(i,j)}(s)}$$

The value $c^{(i,j)}_{min}(u)$, on the other hand, is determined by the set in u's neighborhood with the biggest possible span, i.e., the set that the greedy solution would have picked. Note that this set may not be part of $C_{(i,j)}(u)$. Formally, we have the following:

$$c^{(i,j)}_{min}(u) = \min_{s \in N_u} \frac{1}{d_{(i,j)}(s)}$$

On the other hand, by the definition of $\eta_{(i,j)}(u)$, it holds:

$$c^{(i,j)}_{max}(u) = \eta_{(i,j)}(u) \cdot c^{(i,j)}_{min}(u)$$

Finally, we define the following values:

1. Let $S_{(i,j)}$ be set of candidates that add themselves to S in round i.
2. For each $s \in S_{(i,j)}$ let $U_{(i,j)}(s) \subset N_s$ denote the uncovered elements in s. Note that $d_{(i,j)}(s) := |U_{(i,j)}(s)|$, i.e., the cardinality of $U_{(i,j)}(s)$ is the set's span.
3. Let $U_{(i,j)} := \bigcup_{s \in S_{(i,j)}} U_{(i,j)}(s)$ denote the set of elements that are uncovered at the start of round (i,j) and are covered in this round.

Given these definitions, the number of sets that are added to the solution in round (i,j) can be bounded as follows:

$$|S_{(i,j)}| \leq \sum_{s \in S_{(i,j)}} \frac{d_{(i,j)}(s)}{d_{(i,j)}(s)}$$

As s covers exactly $d_{(i,j)}(s)$ elements:

$$\leq \sum_{s \in S_{(i,j)}} |U_{(i,j)}(s)| \frac{1}{d_{(i,j)}(s)}$$

As $1/d_{(i,j)}(s) \leq c^{(i,j)}_{max}(u)$:

$$\leq \sum_{s \in S_{(i,j)}} \sum_{u \in U_{(i,j)}(s)} c^{(i,j)}_{max}(u)$$

As each $u \in U_{(i,j)}$ is covered by $\mu_{(i,j)}(u)$ sets:

$$= \sum_{u \in U_{(i,j)}} c^{(i,j)}_{max}(u) \cdot \mu_{(i,j)}(u)$$

As $c^{(i,j)}_{max}(u) := \eta_{(i,j)}(u) \cdot c^{(i,j)}_{min}(u)$:

$$\leq \sum_{u \in U_{(i,j)}} c^{(i,j)}_{min}(u) \cdot \eta_{(i,j)}(u) \cdot \mu_{(i,j)}(u)$$

Now we consider the expected value of $|S_{(i,j)}|$. Using the linearity of expectation and the inequality above, we get that:

$$E\left[|S_{(i,j)}|\right] \leq \sum_{u \in V_{\mathcal{U}}} \mathbf{Pr}\big[u \in U_{(i,j)}\big] \cdot \mathbb{E}\left[\mu_{(i,j)}(u) \cdot \eta_{(i,j)}(u) \cdot c_{min}^{(i,j)}(u)\right] \tag{12.1}$$

$$\leq \sum_{u \in V_{\mathcal{U}}} \mathbf{Pr}\big[u \in U_{(i,j)}\big] \cdot \mathbb{E}\left[\mu_{(i,j)}(u)\right] \cdot \mathbb{E}\left[\eta_{(i,j)}(u) \cdot c_{min}^{(i,j)}(u)\right] \tag{12.2}$$

The second line follows from the fact that the random events that determine $\eta_{(i,j)}(u)$ and $\mu_{(i,j)}(u)$ are independent. We will now bound $\mathbb{E}\left[\eta_{(i,j)}(u)c_{min}^{(i,j)}(u)\right]$ and $\mathbb{E}\left[\mu_{(i,j)}(u)\right]$ separately. In particular, for $\mu_{(i,j)}(u)$, we show the following:

Lemma 12.3. *For all elements* $u \in V_{\mathcal{U}}$ *and all round* (i,j)*, it holds:*

$$\mathbb{E}\left[\mu_{(i,j)}(u)\right] \leq 3\Delta^{\frac{1}{k}}$$

The proof is based on the specific properties of the geometric distribution and can be found in Section 12.2.3. It can be regarded as the primary technical contribution of this section as it requires some non-trivial observations on the dependencies between the random decisions made by the algorithm. Similarly, it holds for the difference between the best and the worst set:

Lemma 12.4. *Let* $k \geq 3$*. Then, for every element* u *and round* (i,j) *in which* u *is covered, it holds*

$$\mathbb{E}\left[\eta_{(i,j)}(u)c_{min}^{(i,j)}(u)\right] \leq 25 \cdot \log\Delta \cdot \left(\Delta^{\frac{1}{k}}\right)^2 \mathbb{E}\left[c_{min}^{(i,j)}(u)\right]$$

In contrast to the previous lemma, this statement can be shown via fundamental calculations and elementary combinatorics. The proof can be found in Section 12.2.4. Putting these two insights back in the formula from Equation (12.2), we get:

$$E\left[|S_{(i,j)}|\right] \leq \sum_{u \in V} \mathbf{Pr}\big[u \in U_{(i,j)}\big] \cdot \mathbb{E}\left[\mu_{(i,j)}(u)\right] \mathbb{E}\left[\eta_{(i,j)}(u) \cdot c_{min}^{(i,j)}(u)\right] \tag{12.3}$$

By Lemma 12.3: (12.4)

$$\leq \left(3\Delta^{\frac{1}{k}}\right) \sum_{u \in V} \mathbf{Pr}\big[u \in U_{(i,j)}\big] \cdot \mathbb{E}\left[\eta_{(i,j)}(u) \cdot c_{min}^{(i,j)}(u)\right] \tag{12.5}$$

By Lemma 12.4: (12.6)

$$\leq \left(3\Delta^{\frac{1}{k}}\right) \left(25\left(\sqrt[k]{\Delta}\right)^2 \log\Delta\right) \sum_{u \in V} \mathbf{Pr}\big[u \in U_{(i,j)}\big] \mathbb{E}\left[c_{min}^{(i,j)}(u)\right] \tag{12.7}$$

$$\leq \left(75\Delta^{\frac{3}{k}} \log\Delta\right) \sum_{u \in V} \mathbf{Pr}\big[u \in U_{(i,j)}\big] \mathbb{E}\left[c_{min}^{(i,j)}(u)\right] \tag{12.8}$$

Now recall that each element gets covered eventually, so eventually, each u gets assigned as cost $c_{min}^{(i,j)}(u)$. For each $u \in V_{\mathcal{U}}$ define the expected cost as:

$$\mathbb{E}\left[c_{min}(u)\right] := \sum_{i=0}^{k}\sum_{j=0}^{4k} \mathbf{Pr}\big[u \in U_{(i,j)}\big]\mathbb{E}\left[c_{min}^{(i,j)}(u)\right]$$

Next, we can use the linearity of expectation to sum over all k^2 rounds and get:

$$\begin{aligned}
E\left[|S|\right] &= \sum_{i=0}^{k}\sum_{j=0}^{4k} \mathbb{E}\left[|S_{(i,j)}|\right] && \triangleright \textit{Linearity of Exp.}\\
&\leq \left(75\Delta^{\frac{3}{k}}\log\Delta\right)\sum_{i=0}^{k}\sum_{j=0}^{4k}\sum_{u\in V_{\mathcal{U}}} \mathbf{Pr}\big[u \in U_{(i,j)}\big]\mathbb{E}\left[c_{min}^{(i,j)}(u)\right] && \triangleright \textit{Inequality}\ (12.8)\\
&= \left(75\Delta^{\frac{3}{k}}\log\Delta\right)\sum_{u\in V_{\mathcal{U}}} \mathbb{E}\left[c_{min}(u)\right] && \triangleright \textit{Law of Total Exp.}
\end{aligned}$$

It only remains to bound the *expected* cost assigned to an element when it is covered. By the analysis of the greedy algorithm by Jia et al., it holds that:

Lemma 12.5 (Lemma 3.5 in [JRS02]). *Let $|S_{OPT}|$ be the size of the optimal solution, then it holds:*

$$\sum_{u\in V_{\mathcal{U}}} \mathbb{E}\left[c_{min}(u)\right] \leq H_{\Delta}|S_{OPT}| \leq \log\Delta|S_{OPT}|$$

Here, the term H_{Δ} denotes the Δ^{th} harmonic number.

Proof. Fix any execution $\mathfrak{A}$ of the algorithm, i.e., assign each element a round in which it is covered. We will argue that for all possible executions, the maximal cost of $H_{\Delta}|S_{OPT}|$ is assigned. The proof exploits that c_{min} can always be bounded by the span of a set from the optimal solution.

Consider a set $s \in S_{OPT}$ of degree Δ_s. This set may not be a part solution computed by the algorithm; we only use it to bound the assigned cost. We consider only the rounds in which some neighbor of s is covered in the following. We ignore all the rounds in between. Suppose that there are r distinct rounds $(i_1, j_1), \ldots, (i_r, j_r)$ in which neighbors are covered. To simplify notation, we can enumerate these rounds as $1, \ldots, r$. Note that the concrete indices of the phase and round in which an element is covered do *not* matter for this proof. Further, we can define values $\delta_1, \ldots, \delta_r$ that denote the number of neighbors covered in the corresponding round. To be precise, in the l^{th} round where some elements of s are covered, the algorithm covers exactly $\delta_l \geq 1$ elements. All these values are entirely determined by $\mathfrak{A}$.

In the first round, where any element of s is covered, it is assigned a cost of at most $\frac{1}{\Delta_s}$ (if there is a set with a bigger span, the cost can only get smaller). In the second round, where any neighbor of s is covered, it is assigned a cost of at most $\frac{1}{\Delta_s - \delta_1}$. Recall that δ_1 counts how many elements were covered in the first round. Therefore,

s must have a span of $\Delta_s - \delta_1$ because, per definition, no other neighbor of s was covered in between. This argument can be repeated inductively, such that the total cost after the r^{th} round is bounded by

$$\sum_{u \in N_s} \mathbb{E}\left[c_{min}(u) \mid \mathfrak{A}\right] \leq \sum_{l=1}^{r} \frac{1}{\Delta_s - \delta_l}$$

Seeking an upper bound, let $u_1, \dots, u_{\Delta_s}$ be elements such that u_ℓ is covered before or in the same round as $u_{\ell+1}$. Then, the cost assigned to u_ℓ is bounded by $\frac{1}{\Delta_s-(\ell-1)}$. Thus, the maximal cost that a node of the optimal solution can assign is bounded by

$$\begin{aligned}\sum_{u \in N_s} \mathbb{E}\left[c_{min}(u) \mid \mathfrak{A}\right] &\leq \sum_{l=1}^{r} \frac{1}{\Delta_s - \delta_l} \\ &\leq \sum_{\ell=1}^{\Delta_s} \frac{1}{\Delta_s - (\ell - 1)} \\ &\leq H_{\Delta_s} \leq H_\Delta\end{aligned}$$

Since there are $|S_{OPT}|$ sets in the optimal solution, which all assign cost H_Δ, the lemma follows for assignment $\mathfrak{A}$. As this bound holds for *all* possible executions of the algorithm, the law of total expectation yields the lemma. □

Putting everything together, we get:

$$\begin{aligned}\mathbb{E}\left[|S|\right] &\leq \left(75\Delta^{\frac{3}{k}} \log \Delta\right) \sum_{u \in V_{\mathcal{U}}} \mathbb{E}\left[c_{min}(u)\right] \\ &\leq \left(75\Delta^{\frac{3}{k}} \log \Delta\right) \cdot H_\Delta \cdot |S_{OPT}| && \triangleright \textit{Lemma 12.5} \\ &\leq \left(75\Delta^{\frac{3}{k}} \log \Delta\right) \cdot \log\left(\Delta\right) \cdot |S_{OPT}| \\ &\leq \left(75\Delta^{\frac{3}{k}} \log^2 \Delta\right) \cdot |S_{OPT}| \\ &\in O\left(\Delta^{\frac{3}{k}} \log^2\left(\Delta\right) |S_{OPT}|\right)\end{aligned}$$

This proves the main theorem.

12.2.3 Proof of Lemma 12.3

We will show that for any node $u \in V_{\mathcal{U}}$, it holds that the probability that u is covered by more than t sets simultaneously is exponentially declining in t. In particular, this is independent of the phase and round in which u is covered. The main result of this section is:

Lemma 12.6. *For any value $t > 1$, any element $u \in V$ and any round (i, j) in which u is covered, it holds:*

$$\mathbf{Pr}\left[\mu_{(i,j)}(u) = t\right] \leq \max\{e^{-t/4}, 2\left(1 - 1/\Delta^{\frac{1}{k}}\right)^{t-1}\}$$

We aim to use a result by Miller et al. [MPVX15b] (which needs to be adapted for geometric values). They showed that the number of candidates that pick the earliest possible wake-up time is $\Delta^{\frac{1}{k}}$ in expectation[3]. For each node $u \in V_{\mathcal{U}}$, we consider the *phase* in which u is covered. Recall that every element is *always* covered because in the last round of the last phase, all remaining sets that cover at least one element simply add themselves to the solution.

In the remainder, we will focus only on a single phase i. For readability, we refer to the number of sets that cover u simply as $\mu(u)$ (and not as $\mu_{(i,j)}(u)$) during this proof. Further, since we focus on a single fixed phase, we will just write *round* j instead of *round* (i, j). In particular, when we talk about a round $j \in [0, 4k]$, we always mean *round j of phase i*.

Throughout this proof, we divide the sets into two subsets.

1. First, let N_u be the set of sets in u's neighborhood, i.e., the sets that can cover u. We call these the *neighboring candidates*.

2. Second, let $NN := V_S \setminus N_u$ be all other sets. We call these the *non-neighboring candidates*.

Further, we define the wakeup time Φ_s of each $s \in V_S$ as $\Phi_s := 4k - X_s$. Now we consider the concrete round j in which u is covered. Here, we must distinguish between round $j = 0$ and all other rounds. First, we consider the elements covered in round $j = 0$. In this case, the values X_s of *all* candidates that cover u must be bigger than or equal to $4k$ (before being rounded down). Otherwise, if it held $X_s < 4k$, the value $\Phi_s := 4k - X_s$ would be strictly positive, and the corresponding set would wake up in a later round. In the following, we call a set $s \in V_S$ *early* if it holds $X_s \geq 4k$. Given this definition, it holds:

Lemma 12.7. *Denote $\mathcal{E}_u$ as the number of early sets covering u. Then it holds:*

$$\mathbb{E}\left[\mathcal{E}_u\right] \leq \frac{1}{\Delta^2}$$

Proof. For a single candidate $s \in N_u$, the probability to be *early* is at most

$$\begin{aligned}
\mathbf{Pr}[s \textit{ is early}] &= \sum_{i=0}^{\infty} \mathbf{Pr}[X_s = 4k + i] \\
&= \sum_{i=0}^{\infty} \left(\frac{1}{\Delta^{\frac{1}{k}}}\right)^{4k+i} \left(1 - \frac{1}{\Delta^{\frac{1}{k}}}\right) \\
&\leq \sum_{i=0}^{\infty} \left(\frac{1}{\Delta^{\frac{1}{k}}}\right)^{4k} \left(\frac{1}{\Delta^{\frac{1}{k}}}\right)^{i} \\
&= \frac{1}{\Delta^4} \sum_{i=0}^{\infty} \left(\frac{1}{\Delta^{\frac{1}{k}}}\right)^{i} \leq \frac{2}{\Delta^4}
\end{aligned}$$

[3]Note that that the analysis of a similar result in [GMRV20] only bounds this term in expectation (and not the exact probability) and does not parameterize it on the number of phases as we do. **We need both these aspects for our problem**. Adding these two aspects to their analysis does not seem straightforward as they use a rather complex term to bound the expectation. In contrast, we can exploit the geometric distribution's fundamental properties.

This follows directly from the definition of the geometric distribution. As there are *at most* Δ neighboring candidates that cover u, the expected number of early candidates is at most $\frac{2}{\Delta^3}$ as it holds that:

$$\mathbb{E}\left[\mathcal{E}_u\right] = \sum_{s \in N_u} \mathbf{Pr}[s \textit{ is early}] = \sum_{s \in N_u} \frac{2}{\Delta^4} \leq \frac{2\Delta}{\Delta^4} = \frac{2}{\Delta^3}$$

As $\Delta \geq 2$ this further simplifies to $\frac{1}{\Delta^2}$. □

Given the fact that all set pick their wake-up time independently, the Chernoff bound implies the following:

Lemma 12.8. *Suppose u gets covered in the first round of the phase, then*

$$\mathbf{Pr}[\mu(u) \geq t \mid u \textit{ is covered in round } 0] \leq e^{-\frac{t}{4}}$$

Proof. Since all candidates pick their wake-up time independently and $\mu(u)$ is exactly the number of early candidates, i.e., it holds $\mu(u) = \mathcal{E}_u$, we can use the Chernoff bound (cf. Lemma 1.6) to show that:

$$\begin{aligned}
\mathbf{Pr}[\mu(u) \geq 4t' \mid u \textit{ is covered in round } 0] &= \mathbf{Pr}[\mathcal{E}_u \geq 4t'] \\
&= \mathbf{Pr}\left[\mathcal{E}_u \geq 4t'\Delta^2 \frac{1}{\Delta^2}\right] \\
\text{By Lemma 12.7:} & \\
&\leq \mathbf{Pr}\big[\mathcal{E}_u \geq 4 \cdot t' \cdot \Delta^2 \cdot \mathbb{E}\left[\mathcal{E}_u\right]\big] \\
&\leq \mathbf{Pr}\big[\mathcal{E}_u \geq \left(1 + 3 \cdot t' \cdot \Delta^2\right) \mathbb{E}\left[\mathcal{E}_u\right]\big] \\
\text{Using the Chernoff bound:} & \\
&\leq e^{\frac{-3t' \cdot \Delta^2}{\Delta^2 \cdot 3}} \leq e^{-t'}
\end{aligned}$$

The lemma follows by substituting $t = t'/4$. □

Having dealt with round 0, we now assume that u gets covered in any other round $j > 0$. This part is more complex than the previous one as it requires a more careful analysis of the starting times. First, we will fix all of the random decisions made by the algorithm **except** the decision of the neighboring candidates. In particular, we condition on the following three variables:

1. **The wake-up times of the non-neighboring sets Z_{NN}.**

   It holds $Z_{NN} := (\phi_{s_1}, \ldots, \phi_{s_{|NN|}})$, such that the non-neighboring set $s_l \in NN$ wakes up in round ϕ_{s_l}.

2. **The slots Z_U in which the uncovered elements send a Beep.**

   This random variable contains all random choices made by the uncovered elements. We define it as the set of vectors $Z_U := \{(Y_{u'}^1, \ldots, Y_{u'}^{4k}) \mid u' \in V_{\mathcal{U}}\}$. Here, the variable $Y_{u'}^{\ell}$ denotes whether u' is active and beeps in slot ℓ. Based on this, we can define the *number* of slots in which a set $s \in NN$ receives a Beep from an element in subset $U \subseteq V_{\mathcal{U}}$ as follows:

$$\textsc{Slots}(s, U) := \left|\left\{\ell \in [1, 4k] | \exists u' \in U \cap N_s : Y_{u'}^{\ell} = 1\right\}\right| \tag{12.9}$$

3. **The initial set of uncovered elements $\mathcal{U}_0$.**

   This is the set of uncovered elements in round 0 of phase i. Note that this set is determined in the previous phases. We do not make any statements about its size or the nodes it contains.

We will see that the decision, if a neighboring candidate of u adds itself to the solution, depends on *solely* that candidate's wake-up time if we fix all these choices. Recall that a set adds itself in round j if and only if a) it wakes up in round j **and** b) hears a Beep in at least $3k$ slots of round j. We will show that the random choices of the other nodes define a round $\rho \in [-1, 4k]$ such that in all rounds prior to ρ, s will hear at least $3k$ Beeps and in all later round it will not (and therefore will not add itself). If there is no round where s hears more than $3k$ Beeps, we define $\rho := -1$. Thus, ρ is the last possible round where a candidate may add itself to the solution. If it wakes up after this round, i.e., if it holds $4k - X_s > \rho$, it will not add itself because it will not hear enough Beeps in that round. If it wakes up in or before this round, i.e., if it holds $4k - X_s \leq \rho$, then it wakes up in a round where it receives sufficiently many Beeps. Thus, in the latter case, the concrete round in which it adds itself is *only* determined by the geometric distribution. This intuitive description of ρ is now captured in the following claim:

Claim 19. *Given $(Z_{NN}, Z_U, \mathcal{U}_0)$ for each neighboring set $s \in N_u$ there is a value $\rho_s \in [-1, 4k]$ that denotes the last round in which s can cover u (or takes value $\rho_s = -1$ if there is no such round). Formally, it holds*

$$\mathbf{Pr}[s \textit{ covers u first } \mid \Phi_s > \rho_s] = 0$$

Proof. The idea behind the construction is that the *fixed* values $\mathcal{U}_0$, Z_{NCC}, and Z_U can clearly define the sets $\mathcal{A}_0 \subset V_S$ that add themselves to the solution in round 0. Given Z_{NN}, we see which sets wake up and count the Beeps of their uncovered elements. These Beeps are based on $\mathcal{U}_0$ and Z_U, so we see if enough elements beep in distinct slots. In particular, the set must receive a Beep in at least $3k$ slots, so it must hold that

$$\mathcal{A}_0 := \{s \in NN_u \mid \phi_s = 0 \wedge (|\textsc{Slots}(s, \mathcal{U}_0)| \geq 3k)\}$$

Given $\mathcal{A}_0$, i.e., the sets that add themselves in round 0, we can then compute $\mathcal{U}_1 \subseteq \mathcal{U}_0$, i.e., all uncovered elements in round 1. Then, by repeating the construction, we can use $\mathcal{U}_1$, Z_{NCC}, and Z_U to determine $\mathcal{A}_1$ and therefore $\mathcal{U}_2$ and so on. This can be continued until round τ by the following recursive formulas:

$$\mathcal{A}_t = \{s \in NN_u \mid \phi_s = 0 \wedge (|\textsc{Slots}(s, \mathcal{U}_t)| \geq 3k)\}$$

and

$$\mathcal{U}_t = \left\{ u' \in \mathcal{U} \setminus \{u\} \mid \exists s \in \bigcup_{i=0}^{t-1} \mathcal{A}_i \right\}$$

This yields the uncovered elements $\mathcal{U}_0, \ldots, \mathcal{U}_\tau$. For each candidate $s \in N_u$, we can then clearly identify the last round ρ_s where s receives enough BEEPS to add itself or whether there is no such round. We define:

$$\rho_s := \begin{cases} \arg\max_{0 \leq \tau \leq 4k} \{\textsc{Slots}(s, \mathcal{U}_\tau) \geq 3k\} & \textit{if } \exists \tau \in [0, 4k] : \{\textsc{Slots}(s, \mathcal{U}_\tau) \geq 3k\} \\ -1 & \textit{else} \end{cases}$$

Thus, if the wake-up time is later than ρ_s, s will not hear sufficiently BEEPS. This holds in particular for $\rho_s = -1$ since $\Phi - X_s \geq 0$. □

Given this observation, we can map each outcome of Z_{NCC}, Z_U, and $\mathcal{U}_0$ to a collection of thresholds $\overline{\rho} := (\rho_1, \ldots, \rho_{||N_u})$, one for each neighboring set, with the properties above. If we now condition on $\overline{\rho}$ we get — similar to Miller et al. in [MPVX15b] — that:

Lemma 12.9. *For any possible realization of thresholds $\overline{\rho}$ it holds:*

$$\mathbf{Pr}[\mu(u) = t \mid \overline{\rho}] \leq 2 \left(1 - 1/\Delta^{\frac{1}{k}}\right)^{t-1}$$

Proof. For each neighboring set $s \in N_u$, define the adapted wake-up time as:

$$\Phi'_s := \begin{cases} (\Phi - X_s) & \textit{if } (\Phi - X_s) \leq \rho_s \\ \infty & \textit{else} \end{cases}$$

Here X_s is the geometric random variable drawn to determine the wake-up time. Suppose that u has exactly Δ_u neighboring sets, all of which have a well-defined wake-up time (which may be infinite). We can now order the these adapted wakeup times $\Phi'_{(1)}, \ldots, \Phi'_{(\Delta_u)}$ such that $\Phi'_{(1)}$ is the earliest wakeup and $\Phi'_{(\Delta_u)}$ is the last. For each of these ordered wake-up times $\Phi_{(\ell)}$ we define

- $s_{(\ell)}$ to be the candidate that achieves this time
- $X_{(\ell)}$ is the variable drawn by this set, and
- $\rho_{(\ell)}$ is the threshold.

Given that $\Phi_{(\ell)} \leq \infty$, it holds:

$$\Phi'_{(\ell)} := \Phi - X_{(\ell)}$$

So the variable $X_{(1)}$ is *not* the smallest variable that any neighboring candidate drew, but instead the smallest variable by any neighboring candidate *with finite wake-up time*. Given this definition, we claim the following:

Claim 20. *The candidate $s_{(1)}$ archieving $\Phi'_{(1)}$ covers u if and only if $\Phi'_{(1)} \leq \infty$.*

Proof. This claim can easily be verified by considering the two possibilities. If $\Phi'_{(1)} \leq \infty$, then $\Phi'_{(1)}$ is still the earliest round where some neighborhood candidate tries to add itself and is still a candidate. This follows directly from the definition of $\Phi'_{(1)}$. To be precise, the corresponding set $s_{(1)}$ wakes up in round $\Phi'_{(1)}$ and — since $\Phi'_{(1)} \leq \rho_{(1)}$ — counts more than $3k$ Beeps in this round. Thus, $s_{(1)}$ must cover u as no candidate could have done it before. Otherwise, if $\Phi'_{(1)} = \infty$, then it must holds that $\Phi - X_{(1)} \geq \rho_{(1)}$. Moreover, this implies that $\Phi - X_{(\ell)} \geq \rho_{(\ell)}$ for all other candidates as well. This follows from two facts. First, $\Phi'_{(1)}$ is the smallest wake-up time by definition, so all others must be infinity, too. Second, $\rho_{(\ell)}$ is at most $4k$ and, thus, finite by definition. In this case, *all* potential candidates do not cover enough elements when they wake up. This follows directly from the definition of each $\rho_{(\ell)}$. Thus, in this phase, no candidate will cover u.

□

Thus, for $\mu(u) = t$, the t smallest values must all be equal and non-infinity. Formally:

$$\mathbf{Pr}[\mu(u) = t] = \mathbf{Pr}\Big[\Phi'_{(1)} = \cdots = \Phi'_{(t)} \mid \Phi_{(1)} \leq \infty\Big]$$

Next, we need fundamental calculations based on Miller et al.'s proof for exponential random variables. However, we must adapt them to geometric variables bounded by some maximal value. First, we deal with the bounding. Recall that all of our variables must be smaller than $4k$ (otherwise, u would have been covered in round 0). However, the Law of Total Probability implies:

$$\begin{aligned}
\mathbf{Pr}\big[\mu(u) = t \mid X_{(1)}, \dots, X_{(\Delta_u)} \leq 4k\big] &\leq \frac{\mathbf{Pr}[\mu(u) = t]}{\mathbf{Pr}\big[X_{(1)}, \dots, X_{(\Delta_u)} \leq 4k\big]} \\
\text{As } i \text{ is early iff } X_i \geq 4k\text{:}& \\
&\leq \frac{\mathbf{Pr}[\mu(u) = t]}{1 - \mathbf{Pr}[\mathcal{E}_u \geq 1]} \\
\text{By Lemma 12.7:}& \\
&\leq \frac{\mathbf{Pr}[\mu(u) = t]}{1 - \mathbf{Pr}\big[\mathcal{E}_u \geq \Delta^2 \mathbb{E}\left[\mathcal{E}_u\right]\big]} \\
\text{By Markov's inequality:}& \\
&\leq \frac{\mathbf{Pr}[\mu(u) = t]}{\left(1 - \frac{1}{\Delta^2}\right)} \\
\text{Using } \Delta > 2\text{:}& \\
&\leq 2\mathbf{Pr}[\mu(u) = t]
\end{aligned}$$

Thus, in the following, we will analyze the variables as if they were not bounded. To be precise, we assume that all $X_i \sim Geo(1 - 1/\Delta^{\frac{1}{k}})$. Ultimately, we multiply our final result by 2 to get the desired bound. With this in mind, we show the following claim:

Claim 21. *For any* $0 \leq x \leq \rho_{(\ell)}$, *it holds:*

$$\mathbf{Pr}\left[\Phi'_{(\ell)} < x \mid \Phi'_{(\ell)} \leq x\right] = ({}^{1}/_{\Delta^{1/k}})$$

Proof. Let $x := \rho_{(\ell)} - x'$, then it holds by the definition of $\Phi'_{(\ell)}$:

$$\begin{aligned}\mathbf{Pr}\big[(\Phi - X_{(\ell)}) \leq \rho_{(\ell)} - x'\big] &= \mathbf{Pr}\big[-X_{(\ell)} \leq \rho_{(\ell)} - \Phi - x'\big]\\ &= \mathbf{Pr}\big[X_{(\ell)} \geq (\Phi - \rho_{(\ell)}) + x'\big]\end{aligned}$$

And therefore, for any $x \leq \rho_{(\ell)}$ we see that the wake-up time is only determined by the parameter of the geometric distribution, namely

$$\begin{aligned}&\mathbf{Pr}\left[\Phi'_{(\ell)} < \rho_{(\ell)} - x' \mid \Phi'_{(\ell)} \leq \rho_{(\ell)} - x'\right]\\ =&\mathbf{Pr}\big[(\Phi - X_{(\ell)}) < \rho_{(\ell)} - x' \mid (\Phi - X_{(\ell)}) \leq \rho_{(\ell)} - x'\big]\\ =&\mathbf{Pr}\big[X_{(\ell)} > (\Phi - \rho_{(\ell)}) + x' \mid X_{(\ell)} \geq (\Phi - \rho_{(\ell)}) + x'\big]\end{aligned}$$

Now we can use the fact that the geometric distribution is memoryless and see:

$$\begin{aligned}\mathbf{Pr}\left[\Phi'_{(\ell)} < x \mid \Phi'_{(\ell)} \leq x\right] &= \mathbf{Pr}\big[X_{(\ell)} > (\Phi - \rho_{(\ell)}) + x' \mid X_{(\ell)} \geq (\Phi - \rho_{(\ell)}) + x'\big]\\ \text{Set } y := (\Phi - \rho_{(\ell)}) + x' :&\\ &= \mathbf{Pr}\big[X_{(\ell)} > y \mid X_{(\ell)} \geq y\big]\\ \text{As } X_{(\ell)} \sim Geo(1 - {}^{1}/_{\Delta^{\frac{1}{k}}}) \ :&\\ &= \mathbf{Pr}\big[X_{(\ell)} > 0\big] = ({}^{1}/_{\Delta^{1/k}})\end{aligned}$$

□

Therefore, we can conclude that for two consecutive $\Phi_{(i)}$ and $\Phi_{(i+1)}$, it holds that::

$$\begin{aligned}\mathbf{Pr}\left[\Phi'_{(\ell)} < x \mid \Phi'_{(\ell+1)} = x\right] &= \mathbf{Pr}\left[\Phi'_{(\ell)} < x \mid \Phi'_{(\ell)} \leq x\right] && \triangleright \textit{As } \Phi'_{(\ell)} \leq \Phi'_{(\ell+1)}\\ &= \left(\frac{1}{\Delta^{\frac{1}{k}}}\right)\end{aligned}$$

And thus, for the opposite event, it holds:

$$\mathbf{Pr}\left[\Phi'_{(\ell)} = x \mid \Phi'_{(\ell+1)} = x\right] = \left(1 - \frac{1}{\Delta^{\frac{1}{k}}}\right)$$

Finally, we condition on $\Phi'_{(t)} = \tau$ for a round $0 \leq \tau \leq 4k$. Using the chain rule of conditional probability, we see that:

$$\begin{aligned}
\mathbf{Pr}\Big[\Phi'_{(1)}, \dots, \Phi'_{(t)} = \tau \mid \Phi'_{(t)} = \tau\Big] &= \prod_{i=1}^{t-1} \mathbf{Pr}\Big[\Phi'_{(\ell)} = \tau \mid \Phi'_{(\ell+1)} = \tau, \dots, \Phi'_{(t)} = \tau\Big] \\
&= \prod_{i=1}^{t-1} \mathbf{Pr}\Big[\Phi'_{(\ell)} = \tau \mid \Phi'_{(\ell+1)} = \tau\Big] \\
&\leq \prod_{i=1}^{t-1} \left(1 - \frac{1}{\Delta^{\frac{1}{k}}}\right) = \left(1 - \frac{1}{\Delta^{\frac{1}{k}}}\right)^{t-1}
\end{aligned}$$

Note that this is independent of the actual round, so the law of total probability yields the result. It holds:

$$\begin{aligned}
\mathbf{Pr}[\mu(u) = t] &:= \sum_{\tau=1}^{4k} \mathbf{Pr}\Big[\Phi'_{(t)} = \tau\Big] \mathbf{Pr}\Big[\Phi'_{(1)}, \dots, \Phi'_{(t)} = \tau \mid \Phi'_{(t)} = \tau\Big] \\
&= \sum_{\tau=1}^{4k} \mathbf{Pr}\Big[\Phi'_{(t)} = \tau\Big] \left(1 - \frac{1}{\Delta^{\frac{1}{k}}}\right)^{t-1} = \left(1 - \frac{1}{\Delta^{\frac{1}{k}}}\right)^{t-1}
\end{aligned}$$

As the concrete values of the ρ's are immaterial, the lemma follows by the law of total probability. □

Now we can conclude our proof and calculate the expected value of μ. Given the previous work, the calculation of the expected value is straightforward (as we will show below).

Proof of Lemma 12.3. As implied by the previous lemmas, we split the proof into two parts. First, we consider the *easy case*, when u is covered in the first round of a phase. Here, Lemma 12.7 lets us upper bound the number of additional nodes that also wake up in round 0 as follows:

$$E\left[\mu(u) \mid u \textit{ covered in round o}\right] \leq \frac{1}{\Delta^2} \leq 1.$$

Second, if u is covered in any later round of a phase, we need to use Lemma 12.6. The lemma shows that the number of sets is geometrically distributed (with additional factors). Using the limit of the geometric series, we can show that:

$$\begin{aligned}
E\left[\mu(u) \mid u \textit{ not covered in round o}\right] &\leq \sum_{t=2}^{\infty} t \cdot \mathbf{Pr}[\mu = t \mid u \textit{ not covered in round o}] \\
&\leq \sum_{t=2}^{\infty} 2t \left(1 - {}^{1}\!/\!_{\Delta^{\frac{1}{k}}}\right)^{t-1} = 2\frac{1}{1 - \left(1 - {}^{1}\!/\!_{\Delta^{\frac{1}{k}}}\right)} = 2\Delta^{\frac{1}{k}}
\end{aligned}$$

Finally, the law of total expectation yields that:

$$E\left[\mu(u)\right] \leq E\left[\mu(u) \mid u \textit{ covered in round o}\right] + E\left[\mu(u) \mid u \textit{ not covered in round o}\right] = 3\Delta^{\frac{1}{k}}.$$

This proves the lemma. □

12.2.4 Proof of Lemma 12.4

In the following, we fix an element $u \in V_{\mathcal{U}}$ and suppose it is covered in phase i. As we will see, all rounds of phase i have the same upper bound. To simplify notation, we will not condition on the specific round (of phase i) in which u is covered. We begin our proof by defining two *bad* events:

1. First, we let $\mathcal{B}_1$ be the event that there is any set with span bigger than $6 \cdot \log \Delta \cdot \Delta^{\frac{1}{k}} \cdot \Delta_i$.
2. Second, we let $\mathcal{B}_2$ be the event any set smaller than $\frac{\Delta_i}{4 \cdot \Delta^{\frac{1}{k}}}$ joins the solution.

Recall that our goal in phase i is to add sets that cover roughly Δ_i uncovered elements. So these two events imply that there are sets that greatly deviate from this value. The event $\mathcal{B}_1$ implies that there is a set with too many uncovered neighbors that should have been added earlier, while $\mathcal{B}_2$ implies that there is a set added *too early* because it has too few uncovered neighbors for this phase. In the worst case, both these events hold, and the difference between the sets of the largest and lowest span is huge. This assigns a high cost $\eta_{(i,j)}(u)$ to any element $u \in V_U$ that is covered. On the other hand, if neither of the two events holds, we can bound $\eta_{(i,j)}(u)$ to

$$\mathbb{E}\left[\eta_{(i,j)}(u) \mid \neg\mathcal{B}\right] \leq \frac{6 \cdot \log \Delta \cdot \Delta^{\frac{1}{k}} \cdot \Delta_i}{\Delta_i / 4 \cdot \Delta^{\frac{1}{k}}} = 24 \cdot \log \Delta \cdot \left(\Delta^{\frac{1}{k}}\right)^2.$$

This comes very close to what we want to show. Thus, we want to show that $\eta_{(i,j)}(u)$ is small in expectation. To do this, we define event $\mathcal{B} := \mathcal{B}_1 \cup \mathcal{B}_2$ where both $\mathcal{B}_1$ and $\mathcal{B}_2$ hold and bound its probability. If we can show that $\mathbf{Pr}[\mathcal{B}]$ is small, the lemma follows. In particular, assuming that $\mathbf{Pr}[\mathcal{B}] \leq 2/\Delta$, the law of total expectation implies:

$$\begin{aligned}
&\mathbb{E}\left[\eta_{(i,j)}(u) c_{min}^{(i,j)}(u)\right] \\
=&\mathbf{Pr}[\neg\mathcal{B}] \cdot \mathbb{E}\left[\eta_{(i,j)}(u) c_{min}^{(i,j)}(u) \mid \neg\mathcal{B}\right] + \mathbf{Pr}[\mathcal{B}] \cdot \mathbb{E}\left[\eta_{(i,j)}(u) c_{min}^{(i,j)}(u) \mid \mathcal{B}\right] \\
\leq&24 \cdot \log \Delta \cdot \left(\Delta^{\frac{1}{k}}\right)^2 \mathbb{E}\left[c_{min}^{(i,j)}(u) \mid \neg\mathcal{B}\right] + \frac{2}{\Delta} \cdot \Delta \cdot \mathbb{E}\left[c_{min}^{(i,j)}(u) \mid \mathcal{B}\right] \\
\leq&25 \cdot \log \Delta \cdot \left(\Delta^{\frac{1}{k}}\right)^2 \left(\mathbb{E}\left[\eta_{(i,j)}(u) c_{min}^{(i,j)}(u) \mid \neg\mathcal{B}\right] + \mathbb{E}\left[\eta_{(i,j)}(u) c_{min}^{(i,j)}(u) \mid \mathcal{B}\right]\right) \\
\leq&25 \cdot \log \Delta \cdot \left(\Delta^{\frac{1}{k}}\right)^2 \mathbb{E}\left[c_{min}^{(i,j)}(u)\right]
\end{aligned}$$

Thus, in the following, we show that $\mathbf{Pr}[\mathcal{B}]$ is at most $\frac{2}{\Delta}$. For this, we need some further definitions. Let $H_u^i \subseteq N_u$ denote all neighbors that span more than $6 \cdot \log \Delta \cdot \Delta^{\frac{1}{k}} \cdot \Delta_i$ uncovered elements at any point in phase i and likewise let $L_u^i \subseteq N_u$ denote all neighbors that span less than $\frac{\Delta_i}{4 \cdot \Delta^{\frac{1}{k}}}$ at any point in the phase. Note that it holds

$$\mathbf{Pr}[\mathcal{B}_1] = \mathbf{Pr}\left[\exists s \in H_u^i\right]$$

Further, let S_i denote the sets that add themselves to the solution in phase i. Then, we have:

$$\mathbf{Pr}[\mathcal{B}_2] = \mathbf{Pr}\left[\bigcup_{s \in L_u^i} s \in S_i\right].$$

Then, it holds via union bound that:

$$\mathbf{Pr}[\mathcal{B}] := \mathbf{Pr}[\mathcal{B}_1 \cup \mathcal{B}_2] \leq \mathbf{Pr}[\mathcal{B}_1] + \mathbf{Pr}[\mathcal{B}_2] \leq \mathbf{Pr}\left[\exists s \in H_u^i\right] + \mathbf{Pr}\left[\bigcup_{s \in L_u^i} s \in S_i\right]$$

First, we show that H_u^i is empty with prob. $2/\Delta$. For this, we consider the sets in H_u^i in the previous phase $i-1$. We define $\tilde{H}_u^{i-1}$ as the sets which had

$$6 \cdot \log \Delta \cdot \Delta^{\frac{1}{k}} \cdot \Delta_i := 6 \cdot \log \Delta \cdot \Delta_{i-1}$$

uncovered neighbors when they woke up in phase $i-1$. For these sets, it holds:

Claim 22. *Let $s \in V$ be a set and $c > 6$ be a constant. Suppose that s has at least $c \cdot \log \Delta \cdot \Delta_{i-1}$ uncovered neighbors when it wakes up in phase $i-1$. Then, the probability that s is not added to the solution is smaller than $\Delta^{-(c-2)}$. Formally, it holds:*

$$\mathbf{Pr}\left[s \notin S_{i-1} \mid s \in \tilde{H}_u^{i-1}\right] \leq \frac{1}{\Delta^{(c-2)}}$$

Proof. First, note that for $k < \log \Delta$, it holds $c \cdot \log \Delta \cdot \Delta_{i-1} > 4k$ as we assume $c > 6$. Now consider the round of phase $i-1$ where s woke up. The probability that in a fixed slot (of the $4k$ slots of that round), no elements beeps is at most:

$$\mathbf{Pr}[\textit{No }\textsc{Beep}\textit{ in slot }\ell] \leq \left(1 - \frac{1}{\Delta_{i-1}}\right)^{c \cdot \log \Delta \cdot \Delta_{i-1}} \tag{12.10}$$

$$\leq e^{-c \log \Delta} = \frac{1}{\Delta^c} \tag{12.11}$$

This follows from the fact that s has $c \cdot \log \Delta \cdot \Delta_{i-1}$ uncovered neighbors. Now recall that we initially assumed w.l.o.g. that Δ is bigger than $4k$. Here we finally need this assumption. If $c \geq 6$, a union bound over all k slots then yields that all that with probability higher than $1 - \frac{1}{\Delta^2}$, in every slot, there was at least one active element in the neighborhood of s that beeped. In particular, s counted a Beep in more than $3k$ slots when it woke up. Thus, the set in question must have added itself to the solution in phase $i-1$. Formally, it holds:

$$\mathbf{Pr}[s \in S_{i-1}] \leq \sum_{\ell=1}^{4k} \mathbf{Pr}[\textit{No }\textsc{Beep}\textit{ in slot }\ell] \leq \sum_{\ell=1}^{4k} \frac{1}{\Delta^c} \leq \frac{1}{\Delta^{c-1}}$$

This was to be shown. □

Recall that any set $s \in H_u^i$ must not have joined the solution in phase $i-1$. Suppose that a set has $c \cdot \log\Delta \cdot \Delta^{\frac{1}{k}} \cdot \Delta_i$ uncovered neighbors in phase i. Then it had at least $c \cdot \log\Delta \cdot \Delta_{i-1}$ uncovered neighbors when it woke up in phase $i-1$ as their number can only decrease from phase to phase. So $s \in H_u^i$ directly implies $s \in \tilde{H}_u^{i-1}$. Given the claim from above, the probability that s did not join the solution is

$$\mathbf{Pr}\left[s \notin S_{i-1} \mid s \in \tilde{H}_u^{i-1}\right] \leq \frac{1}{\Delta^{6-4}} \leq \frac{1}{\Delta^2}$$

Here, we used the claim with $c = 6$, which follows from the definition of H_u^i. Thus, we have the desired bound on the sets that span many uncovered elements. Now, we consider a set $s \in L_u^i$ and show that this set is unlikely to add itself to the solution in phase i. We formalize this in the following claim.

Claim 23. *Let s be set with less than $\frac{\Delta_i}{4 \cdot \Delta^{\frac{1}{k}}}$ uncovered elements when it wakes up in phase i. Then, the probability that s joins the solution in phase i is lower than $\frac{1}{\Delta^2}$. Formally, let S_i denote that add themselves to the solution in phase i, then it holds:*

$$\mathbf{Pr}\left[s \in S_i \mid s \in L_u^i\right] \leq \frac{1}{\Delta^2}$$

Proof. Recall that s will eventually wake up in some round phase i but only adds itself to the solution if it hears a Beep in at least $3k$ slots of that round. Obviously, this requires at least $3k$ variables Y_u^ℓ (where u is an uncovered neighbor of s and ℓ is a slot) must be 1. We define:

$$\mathcal{B}_s := \sum_{u \in N_s, C_u = 0} \sum_{\ell=1}^{4k} Y_u^\ell$$

Thus, $\mathcal{B}_s$ is the sum of $4k \cdot \frac{\Delta_i}{4\Delta^{\frac{1}{k}}} < k\Delta$ variables as the have at most $\frac{\Delta_i}{4\Delta^{\frac{1}{k}}}$ uncovered elements and $4k$ rounds. Then, the probability of B_s being bigger than $3k$ is bounded as follows:

$$\mathbf{Pr}[\mathcal{B}_s \geq 3k] \leq \sum_{l=3k}^{k\Delta} \binom{4k \cdot \frac{\Delta_i}{4\Delta^{\frac{1}{k}}}}{l} \left(\frac{1}{\Delta_i}\right)^l$$

$$= \sum_{l=3k}^{k\Delta} \binom{k \cdot \frac{\Delta_i}{\Delta^{\frac{1}{k}}}}{l} \left(\frac{1}{\Delta_i}\right)^l$$

As $\binom{n}{x} < ({en}/{x})^x$

$$\leq \sum_{l=3k}^{k\Delta} \left(\frac{e \cdot k \cdot \Delta_i}{\Delta^{\frac{1}{k}} \cdot l}\right)^l \frac{1}{\Delta_i^l}$$

As $l \geq 3k \geq ek$

$$= \sum_{l=3k}^{k\Delta} \left(\frac{ek}{l}\right)^l \left(\frac{\Delta_i}{\Delta^{\frac{1}{k}}}\right)^l \frac{1}{\Delta_i^l}$$

$$\leq \sum_{l=3k}^{k\Delta} \left(\frac{ek}{3k}\right)^l \left(\frac{\Delta_i}{\Delta^{\frac{1}{k}}}\right)^l \frac{1}{\Delta_i^l}$$

$$\leq \sum_{l=3k}^{k\Delta} \left(\frac{1}{\Delta^{\frac{1}{k}}}\right)^l$$

Setting $l = 3k + (l' - 1)$

$$= \sum_{l'=1}^{k\Delta-3k} \left(\frac{1}{\Delta^3}\right)^{l'} = \left(\frac{1}{\Delta^3}\right) \cdot \sum_{l'=1}^{k\Delta-3k} \left(\frac{1}{\Delta^{\frac{1}{k}}}\right)^{l'-1}$$

As $\sum_{l'=0}^{\infty} a^{l'} \leq \frac{1}{1-a}$

$$\leq \left(\frac{1}{\Delta^3}\right) \cdot \sum_{l'=1}^{\infty} \left(\frac{1}{\Delta^{\frac{1}{k}}}\right)^{l'-1} \leq \left(\frac{1}{\Delta^3}\right) \cdot \sum_{l'=0}^{\infty} \left(\frac{1}{\Delta^{\frac{1}{k}}}\right)^{l'}$$

$$\leq \frac{2}{\Delta^3}$$

As $\Delta \geq 2$

$$\leq \frac{1}{\Delta^2}$$

Thus, the probability that s has beeping neighbors in $3k$ *distinct* slots (and therefore adds itself to the solution) can only be smaller. This proves the claim. □

We now combine our two bounds and see that it holds:

$$\begin{aligned}
\mathbf{Pr}[\mathcal{B}] &\leq \mathbf{Pr}\left[\exists s \in H_u^i\right] + \mathbf{Pr}\left[\bigcup_{s \in L_u^i} s \in S_i\right] \\
&\leq \mathbf{Pr}\left[\bigcup_{s \in H_u^i} s \notin S_{i-1} \mid s \in \tilde{H}_u^{i-1}\right] + \mathbf{Pr}\left[\bigcup_{s \in L_u^i} s \in S_i\right] \\
&\leq \Delta \frac{1}{\Delta^2} + \Delta \frac{1}{\Delta^2} = 2\Delta \frac{1}{\Delta^2} = \frac{2}{\Delta}
\end{aligned}$$

This is what we wanted to show.

12.3 Related Work

Finally, we review some fully distributed approaches for SetCover and related optimization problems. As mentioned before, Jia et al.[JRS02] as well as Kuhn and Wattenhofer[KW03] have presented fast distributed algorithms to solve the DominatingSet problem in time in $O(\log^2 n)$. The goal is to find the minimal set of nodes adjacent to all nodes in the graph $G := (V, E)$. Kuhn et al. further presented an algorithm to solve SetCover (and any covering and packing LP) in $O(\log^2 n)$ rounds in the CONGEST model [KMW06] with logarithmic approximation ratio. This is close to the optimal runtime of $O(\log n)$, which is also proven in that paper. Roughgarden et al. presented a distributed algorithm for convex optimization problems (which includes SetCover) that works on any eventually connected communication network [MRS10]. In particular, it works on dynamic communication networks that can change from round to round. Even though this algorithm is fully distributed, it heavily relies on sequential calculations to cope with these general communication networks. Therefore, its runtime and message complexity are polynomial in the number of nodes. Finally, some works already consider the DominatingSet problem in models tailored to ad-hoc networks. First, there is a paper by Scheideler et al.[SRS08] that considers the SINR model where the nodes are modeled as points in the Euclidean plane. Second, there already is a solution to the DominatingSet-problem in the Beeping-model for unit-disk graphs[YJY$^+$15]. Their algorithm is similar to ours but does not apply to general graphs.

12.4 Conclusion & Future Work

We presented a message-efficient algorithm for the SetCover-problem problem that operates within the BEEPING model and can be extended to solve the $\mathcal{D}$-DominatingSet-problem problem in the CONGEST model. The algorithm achieves an approximation ratio close to the theoretical optimum and due to its simplicity has potential as a versatile building block for tackling other (distributed) optimization problems.

For future work, we envision two promising directions: exploring an extensions of the algorithm to more general optimization problems and addressing the Shortcutting Problem.

Adding Packing or Cost Constraints: In the classical SetCover problem that we considered, each element must be contained in one set of the solution. This definition can be generalized in several ways. For example, one could add an integer δ_u to each set $u \in \mathcal{U}$, s.t., each set u must be contained in δ_u sets. In case of the *uplink problem* that we mentioned in the introduction of this chapter, this variant of the problem may be used to model redundancy. Further, one can imagine a variant of the problem that assigns weights to sets. Each set $s \in \mathcal{S}$ is assigned a positive weight c_s (representing its cost), and the goal is to find the cheapest collection of sets that contain all elements. Note that the classical SetCover corresponds to all sets having a weight of 1.

We believe that, with some engineering, our algorithm can be adapted to handle both these variants of SetCover. Our optimism for this stems from the fact that the algorithm of Jia et al. [JRS02] (which builds the basis for our algorithm) can be extended to solve these variants.

The Shortcutting Problem: The Shortcutting problem is closely related to the *uplink problem* we mentioned in the introduction. It comes in two versions. In both versions, we can assume that we are given a weighted undirected graph $G = (V, E, w)$ and a nonnegative real number δ. In the first version, we are given a parameter k and our goal is to add k shortcut edges of length δ in order to minimize the diameter of the resulting graph. In the second version, we are given a parameter $\mathcal{D}$ and our goal is to add as few shortcut edges of length δ as possible in order to obtain a graph of diameter $\mathcal{D}$.

There is surprisingly little research on this problem, especially in the distributed setting. In the sequential model the problem is understood better, but the research [DZ10, BGP12, FGGM15, LVWX22] is mostly limited to general graphs. Thus, there is much room and opportunity for future work. In particular, we believe that a combination of our SetCover algorithm and the tree covers we computed in the previous chapter has the potential for an efficient and competitive approximation algorithm for restricted graphs that can be implemented in a distributed model.

13 Conclusion to Part II

In this part of the thesis we explored distributed algorithms designed to solve distance-based graph problems. Our goal was to address the challenges of working in distributed computing environments, which require efficient and scalable solutions. Further, we put an emphasis on restricted graph classes like planar graphs, graphs of bounded treewidth, and graphs that exclude a fixed minor because these graph classes not only offer better solutions but also allow for faster algorithms with being to prohibitive for practical applications. We provided algorithms for the fundamental problems of computing low-diameter decompositions and compact routing schemes, which can also act as distance oracles. Our algorithms have near-optimal running time and all established models of distributed computing. In the HYBRID model, which (arguably) captures the behavior of many modern communication networks, we archive polylogarithmic runtimes. In contrast, in the classical CONGEST model, the runtime depends on the hop diameter HD. Both the bounds are optimal, excluding polylogarithmic factors. As HD may be up n, this emphasizes the algorithmic power of adding just a little global communication to a network. Further, the quality of our distributed solutions in terms of approximation guarantees is not far behind their sequential counterparts and/or the provably best quality.

To achieve our results, we massively extend the toolbox for distributed algorithms dealing with restricted graph classes. We believe that these tools are of independent interest for future algorithms. Our new tools include a generic construction of a *weak* path separator. Unlike classical separators, a *weak* separator only separates nodes with respect to the graph's shortest path metric. Our algorithms prove that, in many situations, it is as good as (or at least not significantly worse) a classical separator but can be computed much faster. We believe that there are more graph problems where this is the case. Moreover, we provided an exact analysis of the well-known exponential starting time clustering [MPVX15a, EN16] when used together $(1+\epsilon)$-approximate shortest path computations. This algorithm has been used before, e.g., in [BEL20], but your analysis revealed more nuances in the distribution of the resulting clusters. This allowed even allowed us to improve existing results. Notably, all of our new tools can be implemented with only approximate shortest path computations and minor aggregations. This makes them easily applicable to CONGEST and HYBRID algorithms.

References

[AAC$^+$05] Dana Angluin, James Aspnes, Jiang Chen, Yinghua Wu, and Yitong Yin. Fast Construction of Overlay Networks. In *Proc. of the 17th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 145–154, 2005.

[ABI86] Noga Alon, László Babai, and Alon Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *J. Algorithms*, 7(4):567–583, 1986.

[ABN08] Ittai Abraham, Yair Bartal, and Ofer Neiman. Nearly tight low stretch spanning trees. In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*, pages 781–790. IEEE Computer Society, 2008.

[ACC$^+$20] John Augustine, Keerti Choudhary, Avi Cohen, David Peleg, Sumathi Sivasubramaniam, and Suman Sourav. Distributed graph realizations †. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS), New Orleans, LA, USA, May 18-22, 2020*, pages 158–167. IEEE, 2020.

[AG06] Ittai Abraham and Cyril Gavoille. Object location using path separators. In Eric Ruppert and Dahlia Malkhi, editors, *Proceedings of the Twenty-Fifth Annual ACM Symposium on Principles of Distributed Computing, PODC 2006, Denver, CO, USA, July 23-26, 2006*, pages 188–197. ACM, 2006.

[AGG$^+$14] Ittai Abraham, Cyril Gavoille, Anupam Gupta, Ofer Neiman, and Kunal Talwar. Cops, robbers, and threatening skeletons: padded decomposition for minor-free graphs. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 79–88. ACM, 2014.

[AGG$^+$19a] Ittai Abraham, Cyril Gavoille, Anupam Gupta, Ofer Neiman, and Kunal Talwar. Cops, robbers, and threatening skeletons: Padded decomposition for minor-free graphs. *SIAM J. Comput.*, 48(3):1120–1145, 2019.

[AGG$^+$19b] John Augustine, Mohsen Ghaffari, Robert Gmyr, Kristian Hinnenthal, Fabian Kuhn, Jason Li, and Christian Scheideler. Distributed computation in node-capacitated networks. In *Proc. of the 31st Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2019.

[AGG$^+$19c] John Augustine, Mohsen Ghaffari, Robert Gmyr, Kristian Hinnenthal, Christian Scheideler, Fabian Kuhn, and Jason Li. Distributed computation in node-capacitated networks. In *The 31st ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2019, Phoenix, AZ, USA, June 22-24, 2019*, pages 69–79. ACM, 2019.

[AGGM06] Ittai Abraham, Cyril Gavoille, Andrew V. Goldberg, and Dahlia Malkhi. Routing in networks with low doubling dimension. In *26th IEEE International Conference on Distributed Computing Systems (ICDCS 2006), 4-7 July 2006, Lisboa, Portugal*, page 75. IEEE Computer Society, 2006.

[AGMW10] Ittai Abraham, Cyril Gavoille, Dahlia Malkhi, and Udi Wieder. Strong-diameter decompositions of minor free graphs. *Theory Comput. Syst.*, 47(4):837–855, 2010.

[AHK$^+$20a] John Augustine, Kristian Hinnenthal, Fabian Kuhn, Christian Scheideler, and Philipp Schneider. Shortest paths in a hybrid network model. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1280–1299. SIAM, 2020.

[AHK$^+$20b] John Augustine, Kristian Hinnenthal, Fabian Kuhn, Christian Scheideler, and Philipp Schneider. Shortest paths in a hybrid network model. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1280–1299. SIAM, 2020.

[AKS23] Noga Alon, Michael Krivelevich, and Benny Sudakov. Complete minors and average degree: A short proof. *Journal of Graph Theory*, 103(3):599–602, 2023.

[ALH$^+$23] Ioannis Anagnostides, Christoph Lenzen, Bernhard Haeupler, Goran Zuzic, and Themis Gouleakis. Almost universally optimal distributed laplacian solvers via low-congestion shortcuts. *Distributed Comput.*, 36(4):475–499, 2023.

[AN19] Ittai Abraham and Ofer Neiman. Using petal-decompositions to build a low stretch spanning tree. *SIAM J. Comput.*, 48(2):227–248, 2019.

[AP90] Baruch Awerbuch and David Peleg. Sparse partitions (extended abstract). In *31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, USA, October 22-24, 1990, Volume II*, pages 503–513. IEEE Computer Society, 1990.

[APR$^+$15] John Augustine, Gopal Pandurangan, Peter Robinson, Scott T. Roche, and Eli Upfal. Enabling robust and efficient distributed computation in dynamic peer-to-peer networks. In *Proc. of 56th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 350–369, 2015.

[AS03] James Aspnes and Gauri Shah. Skip graphs. In *Proc. of the 14th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 384–393, 2003.

[AS18] John Augustine and Sumathi Sivasubramaniam. Spartan: A framework for sparse robust addressable networks. In *Proc. of the 32nd IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1060–1069, 2018.

[ASW19] Sepehr Assadi, Xiaorui Sun, and Omri Weinstein. Massively parallel algorithms for finding well-connected components in sparse graphs. In Peter Robinson and Faith Ellen, editors, *Proc. of the 2019 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 461–470. ACM, 2019.

[AW07] James Aspnes and Yinghua Wu. O(logn)-time overlay network construction from graphs with out-degree 1. In Eduardo Tovar, Philippas Tsigas, and Hacène Fouchal, editors, *Proc. of the 11th International Conference on Principles of Distributed Systems (OPODIS)*, volume 4878 of *Lecture Notes in Computer Science*, pages 286–300. Springer, 2007.

[Bar96] Yair Bartal. Probabilistic approximations of metric spaces and its algorithmic applications. In *37th Annual Symposium on Foundations of Computer Science, FOCS '96, Burlington, Vermont, USA, 14-16 October, 1996*, pages 184–193. IEEE Computer Society, 1996.

[Bar98] Yair Bartal. On approximating arbitrary metrices by tree metrics. In Jeffrey Scott Vitter, editor, *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 161–168. ACM, 1998.

[Bar04] Yair Bartal. Graph decomposition lemmas and their role in metric embedding methods. In Susanne Albers and Tomasz Radzik, editors, *Algorithms - ESA 2004, 12th Annual European Symposium, Bergen, Norway, September 14-17, 2004, Proceedings*, volume 3221 of *Lecture Notes in Computer Science*, pages 89–97. Springer, 2004.

[BBD$^+$19] Soheil Behnezhad, Sebastian Brandt, Mahsa Derakhshan, Manuela Fischer, Taghi Hajiaghayi, Mohammad, Richard M Karp, and Jara Uitto. Massively parallel computation of matching and mis in sparse graphs. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 481–490, 2019.

[BBH$^+$21] Alkida Balliu, Sebastian Brandt, Juho Hirvonen, Dennis Olivetti, Mikaël Rabie, and Jukka Suomela. Lower bounds for maximal matchings and maximal independent sets. *J. ACM*, 68(5), December 2021.

[BE10] Leonid Barenboim and Michael Elkin. Sublogarithmic distributed MIS algorithm for sparse graphs using Nash-Williams decomposition. *Distributed Computing*, 22(5-6):363–379, 2010.

[BEGL19] Ruben Becker, Yuval Emek, Mohsen Ghaffari, and Christoph Lenzen. Distributed algorithms for low stretch spanning trees. In Jukka Suomela, editor, *33rd International Symposium on Distributed Computing, DISC 2019, October 14-18, 2019, Budapest, Hungary*, volume 146 of *LIPIcs*, pages 4:1–4:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[BEL20] Ruben Becker, Yuval Emek, and Christoph Lenzen. Low Diameter Graph Decompositions by Approximate Distance Computation. In Thomas Vidick, editor, *11th Innovations in Theoretical Computer Science Conference (ITCS 2020)*, volume 151 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 50:1–50:29, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[BEPS16] Leonid Barenboim, Michael Elkin, Seth Pettie, and Johannes Schneider. The locality of distributed symmetry breaking. *Journal of the ACM*, 63(3):20:1–20:45, 2016.

[BFU19] Sebastian Brandt, Manuela Fischer, and Jara Uitto. Breaking the linear-memory barrier in mpc: Fast mis on trees with strongly sublinear memory. In *International Colloquium on Structural Information and Communication Complexity*, pages 124–138. Springer, 2019.

[BGK$^+$11] Guy E. Blelloch, Anupam Gupta, Ioannis Koutis, Gary L. Miller, Richard Peng, and Kanat Tangwongsan. Near linear-work parallel sdd solvers, low-diameter decomposition, and low-stretch subgraphs. In *Proc. of the 23rd Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, SPAA '11, page 13–22, New York, NY, USA, 2011. Association for Computing Machinery.

[BGP12] Davide Bilò, Luciano Gualà, and Guido Proietti. Improved approximability and non-approximability results for graph diameter decreasing problems. *Theoretical Computer Science*, 417:12–22, 2012. Mathematical Foundations of Computer Science (MFCS 2010).

[BGP13] Andrew Berns, Sukumar Ghosh, and Sriram V. Pemmaraju. Building self-stabilizing overlay networks with the transitive closure framework. *Theor. Comput. Sci.*, 512:2–14, 2013.

[Bod93] H. L. Bodlaender. A tourist guide through treewidth. *Acta Cybernetica*, 11(1-2):1–21, Jan. 1993.

[BS07] Surender Baswana and Sandeep Sen. A simple and linear time randomized algorithm for computing sparse spanners in weighted graphs. *Random Structures & Algorithms*, 30(4):532–563, 2007.

[But14] Vitalik Buterin. Ethereum: A next-generation smart contract and decentralized application platform, 2014. Accessed: 2016-08-22.

[CCF$^+$22] Sam Coy, Artur Czumaj, Michael Feldmann, Kristian Hinnenthal, Fabian Kuhn, Christian Scheideler, Philipp Schneider, and Martijn Struijs. Near-Shortest Path Routing in Hybrid Communication

Networks. In Quentin Bramas, Vincent Gramoli, and Alessia Milani, editors, *25th International Conference on Principles of Distributed Systems (OPODIS 2021)*, volume 217 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 11:1–11:23, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[CCS$^+$22] Sam Coy, Artur Czumaj, Christian Scheideler, Philipp Schneider, and Julian Werthmann. Routing Schemes for Hybrid Communication Networks in Unit-Disk Graphs, October 2022. arXiv:2210.05333 [cs].

[CCS$^+$23] Sam Coy, Artur Czumaj, Christian Scheideler, Philipp Schneider, and Julian Werthmann. Routing schemes for hybrid communication networks. In Sergio Rajsbaum, Alkida Balliu, Joshua J. Daymude, and Dennis Olivetti, editors, *Structural Information and Communication Complexity - 30th International Colloquium, SIROCCO 2023, Alcalá de Henares, Spain, June 6-9, 2023, Proceedings*, volume 13892 of *Lecture Notes in Computer Science*, pages 317–338. Springer, 2023.

[Cha23] Yi-Jun Chang. Efficient distributed decomposition and routing algorithms in minor-free networks and their applications. In Rotem Oshman, Alexandre Nolin, Magnús M. Halldórsson, and Alkida Balliu, editors, *Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing, PODC 2023, Orlando, FL, USA, June 19-23, 2023*, pages 55–66. ACM, 2023.

[Che13] Shiri Chechik. Compact routing schemes with improved stretch. In Panagiota Fatourou and Gadi Taubenfeld, editors, *ACM Symposium on Principles of Distributed Computing, PODC '13, Montreal, QC, Canada, July 22-24, 2013*, pages 33–41. ACM, 2013.

[CHFSV19] Keren Censor-Hillel, Eldar Fischer, Gregory Schwartzman, and Yadu Vasudev. Fast distributed algorithms for testing graph properties. *Distributed Computing*, 32(1):41–57, 2019.

[CHL01] K. A.Wong Chong, Yijie Han, and Tak Wah Lam. Concurrent threads and optimal parallel minimum spanning trees algorithm. *Journal of the ACM*, 48:297–323, 2001.

[CK10] Alejandro Cornejo and Fabian Kuhn. Deploying wireless networks with beeps. In *Distributed Computing, 24th International Symposium, DISC 2010, Cambridge, MA, USA, September 13-15, 2010. Proceedings*, volume 6343 of *Lecture Notes in Computer Science*, pages 148–162. Springer, 2010.

[CKS20] Jannik Castenow, Christina Kolb, and Christian Scheideler. A bounding box overlay for competitive routing in hybrid communication networks. In Nandini Mukherjee and Sriram V. Pemmaraju, editors, *ICDCN 2020: 21st International Conference on Distributed Computing and Networking, Kolkata, India, January 4-7, 2020*, pages 14:1–14:10. ACM, 2020.

[CR20] Philipp Czerner and Harald Räcke. Compact Oblivious Routing in Weighted Graphs, July 2020. arXiv:2007.02427 [cs].

[CS22] Yi-Jun Chang and Hsin-Hao Su. Narrowing the LOCAL-CONGEST gaps in sparse networks via expander decompositions. In Alessia Milani and Philipp Woelfel, editors, *PODC '22: ACM Symposium on Principles of Distributed Computing, Salerno, Italy, July 25 - 29, 2022*, pages 301–312. ACM, 2022.

[CSTW12] Wei Chen, Christian Sommer, Shang-Hua Teng, and Yajun Wang. A compact routing scheme and approximate distance oracle for power-law graphs. *ACM Trans. Algorithms*, 9(1):4:1–4:26, 2012.

[DBB18] Fabien Dufoulon, Janna Burman, and Joffroy Beauquier. Beeping a Deterministic Time-Optimal Leader Election. In *Proceedings of the 32nd International Symposium on Distributed Computing (DISC 2018)*, volume 121 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 20:1–20:17, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[DG10] Emilie Diot and Cyril Gavoille. Path separability of graphs. In Der-Tsai Lee, Danny Z. Chen, and Shi Ying, editors, *Frontiers in Algorithmics, 4th International Workshop, FAW 2010, Wuhan, China, August 11-13, 2010. Proceedings*, volume 6213 of *Lecture Notes in Computer Science*, pages 262–273. Springer, 2010.

[DGH$^+$23] Jinfeng Dou, Thorsten Götte, Henning Hillebrandt, Christian Scheideler, and Julian Werthmann. Brief announcement: Distributed construction of near-optimal compact routing schemes for planar graphs. In Rotem Oshman, Alexandre Nolin, Magnús M. Halldórsson, and Alkida Balliu, editors, *Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing, PODC 2023, Orlando, FL, USA, June 19-23, 2023*, pages 67–70. ACM, 2023.

[DGH$^+$25] Jinfeng Dou, Thorsten Götte, Henning Hillebrandt, Christian Scheideler, and Julian Werthmann. Distributed and parallel low-diameter decompositions for arbitrary and restricted graphs. In Raghu Meka, editor, *16th Innovations in Theoretical Computer Science Conference (ITCS 2025), New York City, NY, USA, January 7–10, 2025*, Leibniz International Proceedings in Informatics (LIPIcs), page (to appear), Dagstuhl, Germany, 2025. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[DGS16] Maximilian Drees, Robert Gmyr, and Christian Scheideler. Churn- and dos-resistant overlay networks based on network reconfiguration. In *Proc. of the 28th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 417–427, 2016.

[Die10] Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer, Heidelberg; New York, fourth edition, 2010.

[DIMV14] Erik D. Demaine, Piotr Indyk, Sepideh Mahabadi, and Ali Vakilian. On streaming and communication complexity of the set cover problem. In *Distributed Computing - 28th International Symposium, DISC 2014, Austin, TX, USA, October 12-15, 2014. Proceedings*, volume 8784 of *Lecture Notes in Computer Science*, pages 484–498. Springer, 2014.

[DP21] Michal Dory and Merav Parter. Fault-tolerant labeling and compact routing schemes. In Avery Miller, Keren Censor-Hillel, and Janne H. Korhonen, editors, *PODC '21: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, July 26-30, 2021*, pages 445–455. ACM, 2021.

[dV23] Tijn de Vos. Minimum cost flow in the CONGEST model. In Sergio Rajsbaum, Alkida Balliu, Joshua J. Daymude, and Dennis Olivetti, editors, *Structural Information and Communication Complexity - 30th International Colloquium, SIROCCO 2023, Alcalá de Henares, Spain, June 6-9, 2023, Proceedings*, volume 13892 of *Lecture Notes in Computer Science*, pages 406–426. Springer, 2023.

[DZ10] Erik D. Demaine and Morteza Zadimoghaddam. Minimizing the diameter of a network using shortcut edges. In *Algorithm Theory - SWAT 2010, 12th Scandinavian Symposium and Workshops on Algorithm Theory, Bergen, Norway, June 21-23, 2010. Proceedings*, pages 420–431, 2010.

[Edm60] J.R. Edmonds. *A Combinatorial Representation for Oriented Polyhedral Surfaces*. University of Maryland, 1960.

[EEST08] Michael Elkin, Yuval Emek, Daniel A. Spielman, and Shang-Hua Teng. Lower-stretch spanning trees. *SIAM J. Comput.*, 38(2):608–628, 2008.

[EFN20] Michael Elkin, Arnold Filtser, and Ofer Neiman. Distributed construction of light networks. In *PODC '20: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, August 3-7, 2020*, pages 483–492. ACM, 2020.

[EN16] Michael Eålkin and Ofer Neiman. On efficient distributed construction of near optimal routing schemes: Extended abstract. In George Giakkoupis, editor, *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing, PODC 2016, Chicago, IL, USA, July 25-28, 2016*, pages 235–244. ACM, 2016.

[EN18a] Michael Elkin and Ofer Neiman. Efficient algorithms for constructing very sparse spanners and emulators. *ACM Transactions on Algorithms (TALG)*, 15(1):1–29, 2018.

[EN18b] Michael Elkin and Ofer Neiman. Near-optimal distributed routing with low memory. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, PODC '18, page 207–216, New York, NY, USA, 2018. Association for Computing Machinery.

[EN19] Michael Elkin and Ofer Neiman. Efficient algorithms for constructing very sparse spanners and emulators. *ACM Trans. Algorithms*, 15(1):4:1–4:29, 2019.

[ENS15] Michael Elkin, Ofer Neiman, and Shay Solomon. Light spanners. *SIAM J. Discret. Math.*, 29(3):1312–1321, 2015.

[FGGM15] Fabrizio Frati, Serge Gaspers, Joachim Gudmundsson, and Luke Mathieson. Augmenting graphs to minimize the diameter. *Algorithmica*, 72(4):995–1010, 2015.

[FGS19] Michael Feldmann, Thorsten Götte, and Christian Scheideler. A loosely self-stabilizing protocol for randomized congestion control with logarithmic memory. In Mohsen Ghaffari, Mikhail Nesterenko, Sébastien Tixeuil, Sara Tucci, and Yukiko Yamauchi, editors, *Stabilization, Safety, and Security of Distributed Systems - 21st International Symposium, SSS 2019, Pisa, Italy, October 22-25, 2019, Proceedings*, volume 11914 of *Lecture Notes in Computer Science*, pages 149–164. Springer, 2019.

[FHS20] Michael Feldmann, Kristian Hinnenthal, and Christian Scheideler. Fast hybrid network algorithms for shortest paths in sparse graphs. In *Proceedings of the 24th International Conference on Principles of Distributed Systems (OPODIS)*, pages 31:1–31:16, 2020.

[Fil19] Arnold Filtser. On Strong Diameter Padded Decompositions. In Dimitris Achlioptas and László A. Végh, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2019)*, volume 145 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 6:1–6:21, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[Flo62] Robert W. Floyd. Algorithm 97: Shortest path. *Commun. ACM*, 5(6):345, 1962.

[FN22] Arnold Filtser and Ofer Neiman. Light spanners for high dimensional norms via stochastic decompositions. *Algorithmica*, 84(10):2987–3007, 2022.

[FPR$^+$10] Nathan Farrington, George Porter, Sivasankar Radhakrishnan, Hamid Hajabdolali Bazzaz, Vikram Subramanya, Yeshaiahu Fainman, George Papen, and Amin Vahdat. Helios: a hybrid electrical/optical switch architecture for modular data centers. In *Proceedings of the ACM SIGCOMM 2010 Conference*, pages 339–350, 2010.

[FRT04] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *J. Comput. Syst. Sci.*, 69(3):485–497, 2004.

[FSS20] Michael Feldmann, Christian Scheideler, and Stefan Schmid. Survey on algorithms for self-stabilizing overlay networks. *ACM Computing Surveys*, 53(4), 2020.

[FT03] Jittat Fakcharoenphol and Kunal Talwar. An Improved Decomposition Theorem for Graphs Excluding a Fixed Minor. In Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, Sanjeev Arora, Klaus Jansen, José D. P. Rolim, and Amit Sahai, editors, *Approximation, Randomization, and Combinatorial Optimization.. Algorithms and Techniques*, volume 2764, pages 36–46. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003. Series Title: Lecture Notes in Computer Science.

[FV18] Hendrik Fichtenberger and Yadu Vasudev. A two-sided error distributed property tester for conductance. In *43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[GG23] Mohsen Ghaffari and Christoph Grunau. Faster deterministic distributed MIS and approximate matching. In Barna Saha and Rocco A. Servedio, editors, *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 1777–1790. ACM, 2023.

[GG24] Mohsen Ghaffari and Christoph Grunau. Near-optimal deterministic network decomposition and ruling set, and improved MIS. In *65th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2024, Chicago, IL, USA, October 27-30, 2024*, pages 2148–2179. IEEE, 2024.

[GGJ20] Mohsen Ghaffari, Christoph Grunau, and Ce Jin. Improved MPC Algorithms for MIS, Matching, and Coloring on Trees and Beyond. In Hagit Attiya, editor, *34th International Symposium on Distributed Computing (DISC 2020)*, volume 179 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 34:1–34:18, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

[GGK$^+$18] Mohsen Ghaffari, Themis Gouleakis, Christian Konrad, Slobodan Mitrović, and Ronitt Rubinfeld. Improved massively parallel computation algorithms for mis, matching, and vertex cover. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, pages 129–138, 2018.

[GGR21] Mohsen Ghaffari, Christoph Grunau, and Václav Rozhon. Improved deterministic network decomposition. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 2904–2923. SIAM, 2021.

[GH16a] Mohsen Ghaffari and Bernhard Haeupler. Distributed algorithms for planar networks I: planar embedding. In George Giakkoupis, editor, *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing, PODC 2016, Chicago, IL, USA, July 25-28, 2016*, pages 29–38. ACM, 2016.

[GH16b] Mohsen Ghaffari and Bernhard Haeupler. Distributed algorithms for planar networks II: low-congestion shortcuts, mst, and min-cut. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 202–219. SIAM, 2016.

[GH21] Mohsen Ghaffari and Bernhard Haeupler. Low-congestion shortcuts for graphs excluding dense minors. In Avery Miller, Keren Censor-Hillel, and Janne H. Korhonen, editors, *PODC '21: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, July 26-30, 2021*, pages 213–221. ACM, 2021.

[Gha16] Mohsen Ghaffari. An improved distributed algorithm for maximal independent set. In *Proceedings of the 27th annual ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 270–277. SIAM, 2016.

[Gha19] Mohsen Ghaffari. Distributed maximal independent set using small messages. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '19, page 805–820. Society for Industrial and Applied Mathematics, 2019.

[GHS19a] Thorsten Götte, Kristian Hinnenthal, and Christian Scheideler. Faster construction of overlay networks. In Keren Censor-Hillel and Michele Flammini, editors, *Structural Information and Communication Complexity - 26th International Colloquium, SIROCCO 2019, L'Aquila, Italy, July 1-4, 2019, Proceedings*, volume 11639 of *Lecture Notes in Computer Science*, pages 262–276. Springer, 2019.

[GHS19b] Thorsten Götte, Kristian Hinnenthal, and Christian Scheideler. Faster construction of overlay networks. In *International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pages 262–276. Springer, 2019.

[GHSS17] Robert Gmyr, Kristian Hinnenthal, Christian Scheideler, and Christian Sohler. Distributed Monitoring of Network Properties: The Power of Hybrid Networks. In *Proc. of the 44th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 137:1–137:15, 2017.

[GHSW21] Thorsten Götte, Kristian Hinnenthal, Christian Scheideler, and Julian Werthmann. Time-optimal construction of overlay networks. In Avery Miller, Keren Censor-Hillel, and Janne H. Korhonen, editors, *PODC '21: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, July 26-30, 2021*, pages 457–468. ACM, 2021.

[GHSW23] Thorsten Götte, Kristian Hinnenthal, Christian Scheideler, and Julian Werthmann. Time-optimal construction of overlay networks. *Distributed Comput.*, 36(3):313–347, 2023.

[GHZ21] Mohsen Ghaffari, Bernhard Haeupler, and Goran Zuzic. Hop-constrained oblivious routing. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1208–1220, Virtual Italy, June 2021. ACM.

[GKK$^+$18] Mohsen Ghaffari, Andreas Karrenbauer, Fabian Kuhn, Christoph Lenzen, and Boaz Patt-Shamir. Near-optimal distributed maximum flow. *SIAM J. Comput.*, 47(6):2078–2117, 2018.

[GKSW21] Thorsten Götte, Christina Kolb, Christian Scheideler, and Julian Werthmann. Beep-and-sleep: Message and energy efficient set cover. In Leszek Gasieniec, Ralf Klasing, and Tomasz Radzik, editors, *Algorithms for Sensor Systems - 17th International Symposium on Algorithms and Experiments for Wireless Sensor Networks, ALGOSENSORS 2021, Lisbon, Portugal, September 9-10, 2021, Proceedings*, volume 12961 of *Lecture Notes in Computer Science*, pages 94–110. Springer, 2021.

[GKSW23] Thorsten Götte, Christina Kolb, Christian Scheideler, and Julian Werthmann. Beep-and-sleep: Message and energy efficient set cover. *Theor. Comput. Sci.*, 950:113756, 2023.

[GMRV20] Christoph Grunau, Slobodan Mitrovic, Ronitt Rubinfeld, and Ali Vakilian. Improved local computation algorithm for set cover via sparsification. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2993–3011. SIAM, 2020.

[GMS04] Christos Gkantsidis, Milena Mihail, and Amin Saberi. Random walks in peer-to-peer networks. In *Proc. of the 23rd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*. IEEE, 2004.

[GP16] Mohsen Ghaffari and Merav Parter. Mst in log-star rounds of congested clique. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*, pages 19–28, 2016.

[GP17] Mohsen Ghaffari and Merav Parter. Near-optimal distributed DFS in planar graphs. In Andréa W. Richa, editor, *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*, volume 91 of *LIPIcs*, pages 21:1–21:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[GPRT20] Seth Gilbert, Gopal Pandurangan, Peter Robinson, and Amitabh Trehan. Dconstructor: Efficient and robust network construction with polylogarithmic overhead. In *Proc. of ACM Symposium on Principles of Distributed Computing (PODC)*, pages 438–447. ACM, 2020.

[GS22] Thorsten Götte and Christian Scheideler. Brief announcement: The (limited) power of multiple identities: Asynchronous byzantine reliable broadcast with improved resilience through collusion. In Kunal Agrawal and I-Ting Angelina Lee, editors, *SPAA '22: 34th ACM Symposium on Parallelism in Algorithms and Architectures, Philadelphia, PA, USA, July 11 - 14, 2022*, pages 99–101. ACM, 2022.

[GSS18] Thorsten Götte, Christian Scheideler, and Alexander Setzer. On underlay-aware self-stabilizing overlay networks. In Taisuke Izumi and Petr Kuznetsov, editors, *Stabilization, Safety, and Security of Distributed Systems - 20th International Symposium, SSS 2018, Tokyo, Japan, November 4-7, 2018, Proceedings*, volume 11201 of *Lecture Notes in Computer Science*, pages 50–64. Springer, 2018.

[GVS19a] Thorsten Götte, Vipin Ravindran Vijayalakshmi, and Christian Scheideler. Always be two steps ahead of your enemy. In *2019 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2019, Rio de Janeiro, Brazil, May 20-24, 2019*, pages 1073–1082. IEEE, 2019.

[GVS19b] Thorsten Götte, Vipin Ravindran Vijayalakshmi, and Christian Scheideler. Always be Two Steps Ahead of Your Enemy. In *Proc. of the 33rd IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2019.

[GZ22a] Mohsen Ghaffari and Goran Zuzic. Universally-Optimal Distributed Exact Min-Cut. In *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing*, pages 281–291, Salerno Italy, July 2022. ACM.

[GZ22b] Mohsen Ghaffari and Goran Zuzic. Universally-Optimal Distributed Exact Min-Cut. In *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing*, pages 281–291, Salerno Italy, July 2022. ACM.

[HKP$^+$11] Daniel Halperin, Srikanth Kandula, Jitendra Padhye, Paramvir Bahl, and David Wetherall. Augmenting data center networks with multi-gigabit wireless links. In *Proceedings of the ACM SIGCOMM 2011 conference*, pages 38–49, 2011.

[HL18] Bernhard Haeupler and Jason Li. Faster distributed shortest path approximations via shortcuts. In Ulrich Schmid and Josef Widder, editors, *32nd International Symposium on Distributed Computing, DISC 2018, New Orleans, LA, USA, October 15-19, 2018*, volume 121 of *LIPIcs*, pages 33:1–33:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[HRG22] Bernhard Haeupler, Harald Räcke, and Mohsen Ghaffari. Hop-constrained expander decompositions, oblivious routing, and distributed universal optimality. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 1325–1338. ACM, 2022.

[HWZ21] Bernhard Haeupler, David Wajc, and Goran Zuzic. Universally-optimal distributed algorithms for known topologies. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 1166–1179. ACM, 2021.

[HZ01] Shay Halperin and Uri Zwick. Optimal Randomized EREW PRAM Algorithms for Finding Spanning Forests. *Journal of Algorithms*, 39(1):1–46, 2001.

[IEWM23] Taisuke Izumi, Yuval Emek, Tadashi Wadayama, and Toshimitsu Masuzawa. Deterministic fault-tolerant connectivity labeling scheme. In Rotem Oshman, Alexandre Nolin, Magnús M. Halldórsson, and Alkida Balliu, editors, *Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing, PODC 2023, Orlando, FL, USA, June 19-23, 2023*, pages 190–199. ACM, 2023.

[IKNS22] Taisuke Izumi, Naoki Kitamura, Takamasa Naruse, and Gregory Schwartzman. Fully polynomial-time distributed computation in low-treewidth graphs. In Kunal Agrawal and I-Ting Angelina Lee, editors, *SPAA '22: 34th ACM Symposium on Parallelism in Algorithms and Architectures, Philadelphia, PA, USA, July 11 - 14, 2022*, pages 11–22. ACM, 2022.

[IMR$^+$17] Piotr Indyk, Sepideh Mahabadi, Ronitt Rubinfeld, Jonathan R. Ullman, Ali Vakilian, and Anak Yodpinyanee. Fractional set cover in the streaming model. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2017, August 16-18, 2017, Berkeley, CA, USA*, volume 81 of *LIPIcs*, pages 12:1–12:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[IMR$^+$18] Piotr Indyk, Sepideh Mahabadi, Ronitt Rubinfeld, Ali Vakilian, and Anak Yodpinyanee. Set cover in sub-linear time. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 2467–2486. SIAM, 2018.

[IV19] Piotr Indyk and Ali Vakilian. Tight trade-offs for the maximum k-coverage problem in the general streaming model. In *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, pages 200–217. ACM, 2019.

[JKSS18] Daniel Jung, Christina Kolb, Christian Scheideler, and Jannik Sundermeier. Competitive routing in hybrid communication networks. In *Algorithms for Sensor Systems - 14th International Symposium on Algorithms and Experiments for Wireless Sensor Networks, ALGOSENSORS 2018, Helsinki, Finland, August 23-24, 2018, Revised Selected Papers*, pages 15–31, 2018.

[JN18] Tomasz Jurdziński and Krzysztof Nowicki. Mst in $o(1)$ rounds of congested clique. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2620–2632. SIAM, 2018.

[JRS02] Lujun Jia, Rajmohan Rajaraman, and Torsten Suel. An efficient distributed algorithm for constructing small dominating sets. *Distributed Comput.*, 15(4):193–205, 2002.

[JRS$^+$14] Riko Jacob, Andréa W. Richa, Christian Scheideler, Stefan Schmid, and Hanjo Täubig. Skip+: A self-stabilizing skip graph. *Journal of the ACM*, 61(6):36:1–36:26, 2014.

[Kar00] David R Karger. Minimum cuts in near-linear time. *Journal of the ACM (JACM)*, 47(1):46–76, 2000.

[KL14] Tsz Chiu Kwok and Lap Chi Lau. Lower bounds on expansions of graph powers. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2014)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2014.

[KL24] Tuukka Korhonen and Daniel Lokshtanov. Induced-minor-free graphs: Separator theorem, subexponential algorithms, and improved hardness of recognition. In David P. Woodruff, editor, *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*, pages 5249–5275. SIAM, 2024.

[Kle24] Klein, Philip and Mozes, Shay. Optimization algorithms for planar graphs. https://planarity.org, 2024. [Online; accessed 14-June-2024].

[KMW04] Fabian Kuhn, Thomas Moscibroda, and Rogert Wattenhofer. What cannot be computed locally! In *Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, pages 300–309, 2004.

[KMW06] Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. The price of being near-sighted. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, Florida, USA, January 22-26, 2006*, pages 980–989. ACM Press, 2006.

[Kol22] Christina Kolb. *Competitive routing in hybrid communication networks and message efficient SetCover in Ad Hoc networks*. PhD thesis, University of Paderborn, Germany, 2022.

[KPR93] Philip N. Klein, Serge A. Plotkin, and Satish Rao. Excluded minors, network decomposition, and multicommodity flow. In S. Rao Kosaraju, David S. Johnson, and Alok Aggarwal, editors, *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pages 682–690. ACM, 1993.

[KS22] Fabian Kuhn and Philipp Schneider. Routing Schemes and Distance Oracles in the Hybrid Model. In Christian Scheideler, editor, *36th International Symposium on Distributed Computing (DISC 2022)*, volume 246 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 28:1–28:22, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[KST13] Ken-ichi Kawarabayashi, Christian Sommer, and Mikkel Thorup. More compact oracles for approximate distances in undirected planar graphs. In Sanjeev Khanna, editor, *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 550–563. SIAM, 2013.

[KTT93] Ming-Yang Kao, Shang-Hua Teng, and Kentaro Toyama. Improved parallel depth-first search in undirected planar graphs. In G. Goos, J. Hartmanis, Frank Dehne, Jörg-Rüdiger Sack, Nicola Santoro, and Sue Whitesides, editors, *Algorithms and Data Structures*, volume 709, pages 409–420. Springer Berlin Heidelberg, Berlin, Heidelberg, 1993. Series Title: Lecture Notes in Computer Science.

[KW03] Fabian Kuhn and Roger Wattenhofer. Constant-time distributed dominating set approximation. In *Proceedings of the Twenty-Second ACM Symposium on Principles of Distributed Computing, PODC 2003, Boston, Massachusetts, USA, July 13-16, 2003*, pages 25–32. ACM, 2003.

[LGT14] James R Lee, Shayan Oveis Gharan, and Luca Trevisan. Multiway spectral partitioning and higher-order cheeger inequalities. *Journal of the ACM (JACM)*, 61(6):1–30, 2014.

[ŁMOS20] Jakub Łącki, Slobodan Mitrović, Krzysztof Onak, and Piotr Sankowski. Walking randomly, massively, and efficiently. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 364–377, 2020.

[LMR21] Reut Levi, Moti Medina, and Dana Ron. Property testing of planarity in the CONGEST model. *Distributed Comput.*, 34(1):15–32, 2021.

[LP13] Christoph Lenzen and Boaz Patt-Shamir. Fast routing table construction using small messages: extended abstract. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 381–390, 2013.

[LP15] Christoph Lenzen and Boaz Patt-Shamir. Fast partial distance estimation and applications. In Chryssis Georgiou and Paul G. Spirakis, editors, *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21 - 23, 2015*, pages 153–162. ACM, 2015.

[LP19] Jason Li and Merav Parter. Planar diameter via metric compression. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 152–163. ACM, 2019.

[LPP19] Christoph Lenzen, Boaz Patt-Shamir, and David Peleg. Distributed distance computation and routing with small messages. *Distributed Comput.*, 32(2):133–157, 2019.

[LPS24] Yaowei Long, Seth Pettie, and Thatchaphol Saranurak. Connectivity labeling schemes for edge and vertex faults via expander hierarchies. *CoRR*, abs/2410.18885, 2024.

[LS90] László Lovász and Miklós Simonovits. The mixing rate of markov chains, an isoperimetric inequality, and computing the volume. In *Proc. of 31st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 346–354. IEEE, 1990.

[LS03] Ching Law and Kai-Yeung Siu. Distributed construction of random expander networks. In *Proceedings IEEE INFOCOM 2003, The 22nd Annual Joint Conference of the IEEE Computer and Communications Societies, San Franciso, CA, USA, March 30 - April 3, 2003*, pages 2133–2143. IEEE Computer Society, 2003.

[LSZ20] Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. Efficient graph minors theory and parameterized algorithms for (planar) disjoint paths. In Fedor V. Fomin, Stefan Kratsch, and Erik Jan van Leeuwen, editors, *Treewidth, Kernels, and Algorithms - Essays Dedicated to Hans L. Bodlaender on the Occasion of His 60th Birthday*, volume 12160 of *Lecture Notes in Computer Science*, pages 112–128. Springer, 2020.

[LT80] Richard J. Lipton and Robert Endre Tarjan. Applications of a planar separator theorem. *SIAM J. Comput.*, 9(3):615–627, 1980.

[LTZ20] Sixue Cliff Liu, Robert E. Tarjan, and Peilin Zhong. Connected components on a PRAM in log diameter time. In Christian Scheideler and Michael Spear, editors, *Proc. of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), Virtual Event, USA, July 15-17, 2020*, pages 359–369. ACM, 2020.

[Lub86] Michael Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM journal on computing*, 15(4):1036–1053, 1986.

[LVWX22] Kevin Lu, Virginia Vassilevska Williams, Nicole Wein, and Zixuan Xu. Better lower bounds for shortcut sets and additive spanners via an improved alternation product. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 3311–3331. SIAM, 2022.

[MPVX15a] Gary L. Miller, Richard Peng, Adrian Vladu, and Shen Chen Xu. Improved parallel algorithms for spanners and hopsets. In *Proc. of the 27th ACM symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 192–201, 2015.

[MPVX15b] Gary L. Miller, Richard Peng, Adrian Vladu, and Shen Chen Xu. Improved parallel algorithms for spanners and hopsets. In *Proceedings of the 27th ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2015, Portland, OR, USA, June 13-15, 2015*, pages 192–201. ACM, 2015.

[MPX13] Gary L. Miller, Richard Peng, and Shen Chen Xu. Parallel graph decompositions using random shifts. In Guy E. Blelloch and Berthold Vöcking, editors, *25th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '13, Montreal, QC, Canada - July 23 - 25, 2013*, pages 196–203. ACM, 2013.

[MRNZ09] Yves Métivier, John Michael Robson, Saheb-Djahromi Nasser, and Akka Zemmari. An optimal bit complexity randomized distributed mis algorithm. In *International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pages 323–337. Springer, 2009.

[MRS10] Damon Mosk-Aoyama, Tim Roughgarden, and Devavrat Shah. Fully distributed algorithms for convex optimization problems. *SIAM J. Optim.*, 20(6):3260–3279, 2010.

[MS06] Damon Mosk-Aoyama and Devavrat Shah. Computing separable functions via gossip. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Principles of Distributed Computing, PODC 2006, Denver, CO, USA, July 23-26, 2006*, pages 113–122, 2006.

[MT01] B. Mohar and C. Thomassen. *Graphs on Surfaces*. Johns Hopkins Studies in Nineteenth C Architecture Series. Johns Hopkins University Press, 2001.

[MU05] Michael Mitzenmacher and Eli Upfal. *Probability and Computing*. Cambridge University Press, 2005.

[MW20] Wolfgang Mulzer and Max Willert. Compact routing in unit disk graphs. In Yixin Cao, Siu-Wing Cheng, and Minming Li, editors, *31st International Symposium on Algorithms and Computation, ISAAC 2020, December 14-18, 2020, Hong Kong, China (Virtual Conference)*, volume 181 of *LIPIcs*, pages 16:1–16:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, Dec 2008. Accessed: 2015-07-01.

[Now19] Krzysztof Nowicki. A deterministic algorithm for the mst problem in constant rounds of congested clique. *arXiv preprint arXiv:1912.04239*, 2019.

[NW64] C. St.J. A. Nash-Williams. Decomposition of finite graphs into forests. *Journal of the London Mathematical Society*, s1-39(1):12–12, 1964.

[OG02] Scott Oaks and Li Gong. *Jxta in a Nutshell*. O ' Reilly & Associates, Inc., USA, 2002.

[Pel00] David Peleg. *Distributed Computing: A Locality-Sensitive Approach*. Society for Industrial and Applied Mathematics, USA, 2000.

[PPP24] Merav Parter, Asaf Petruschka, and Seth Pettie. Connectivity labeling and routing with multiple vertex failures. In Bojan Mohar, Igor Shinkar, and Ryan O'Donnell, editors, *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024, Vancouver, BC, Canada, June 24-28, 2024*, pages 823–834. ACM, 2024.

[PR02] Seth Pettie and Vijaya Ramachandran. A randomized time-work optimal parallel algorithm for finding a minimum spanning forest. *SIAM Journal on Computing*, 31:1879–1895, 2002.

[PU89] David Peleg and Eli Upfal. A trade-off between space and efficiency for routing tables. *J. ACM*, 36(3):510–530, 1989.

[Rac02] H. Racke. Minimizing congestion in general networks. In *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.*, pages 43–52, Vancouver, BC, Canada, 2002. IEEE Comput. Soc.

[REGH22] Václav Rozhon, Michael Elkin, Christoph Grunau, and Bernhard Haeupler. Deterministic low-diameter decompositions for weighted graphs and distributed and parallel applications. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*, pages 1114–1121. IEEE, 2022.

[RG20] Václav Rozhon and Mohsen Ghaffari. Polylogarithmic-time deterministic network decomposition and distributed derandomization. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 350–363. ACM, 2020.

[RGH$^+$22] Václav Rozhon, Christoph Grunau, Bernhard Haeupler, Goran Zuzic, and Jason Li. Undirected $(1+\epsilon)$-shortest paths via minor-aggregates: near-optimal deterministic parallel and distributed algorithms. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 478–487. ACM, 2022.

[RR89] Vijaya Ramachandran and John H. Reif. An optimal parallel algorithm for graph planarity (extended abstract). In *30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, North Carolina, USA, 30 October - 1 November 1989*, pages 282–287. IEEE Computer Society, 1989.

[RS86] Neil Robertson and Paul D. Seymour. Graph minors. II. algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986.

[RS03] Neil Robertson and Paul D Seymour. Graph minors. xvi. excluding a non-planar graph. *Journal of Combinatorial Theory, Series B*, 89(1):43–76, 2003.

[Rä08] Harald Räcke. Optimal hierarchical decompositions for congestion minimization in networks. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 255–264, Victoria British Columbia Canada, May 2008. ACM.

[Rä09] Harald Räcke. Survey on Oblivious Routing Strategies. In Klaus Ambos-Spies, Benedikt Löwe, and Wolfgang Merkle, editors, *Mathematical Theory and Computational Practice*, volume 5635, pages 419–429. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. Series Title: Lecture Notes in Computer Science.

[Sch00] Christian Scheideler. *Probabilistic Methods for Coordination Problems*. Habilitation, Universität Paderborn, Heinz Nixdorf Institut, Theoretische Informatik, 2000. ISBN 3-931466-77-9.

[Sch23a] Philipp Schneider. *Power and limitations of hybrid communication networks*. PhD thesis, University of Freiburg, Freiburg im Breisgau, Germany, 2023.

[Sch23b] Jonas Stephan Schweichhart. *Minimum Edge Cuts in Overlay Networks*. Master's thesis, Paderborn University, Paderborn, Germany, March 2023.

[Sin12] Alistair Sinclair. *Algorithms for random generation and counting: a Markov chain approach*. Springer Science & Business Media, 2012.

[SJ89] Alistair Sinclair and Mark Jerrum. Approximate counting, uniform generation and rapidly mixing markov chains. *Inf. Comput.*, 82(1):93–133, 1989.

[SMPU13] Atish Das Sarma, Anisur Rahaman Molla, Gopal Pandurangan, and Eli Upfal. Fast distributed pagerank computation. In *International Conference on Distributed Computing and Networking*, pages 11–26. Springer, 2013.

[SR06] Daniel Stutzbach and Reza Rejaie. Understanding churn in peer-to-peer networks. IMC '06, page 189–202, New York, NY, USA, 2006. Association for Computing Machinery.

[SRS08] Christian Scheideler, Andréa W. Richa, and Paolo Santi. An o(log n) dominating set protocol for wireless ad-hoc networks under the physical interference model. In *Proceedings of the 9th ACM Interational Symposium on Mobile Ad Hoc Networking and Computing, MobiHoc 2008, Hong Kong, China, May 26-30, 2008*, pages 91–100. ACM, 2008.

[TBKC18] Anis Tell, Wale Babalola, George Kalaba Kalebaila, and Krishna C. Chinta. Sd-wan: A modern hybrid-wan to enable digital transformation for businesses. 2018.

[Tho04] Mikkel Thorup. Compact oracles for reachability and approximate distances in planar digraphs. *J. ACM*, 51(6):993–1024, nov 2004.

[TV85] Robert E Tarjan and Uzi Vishkin. An efficient parallel biconnectivity algorithm. *SIAM Journal on Computing*, 14(4):862–874, 1985.

[TZ01] Mikkel Thorup and Uri Zwick. Compact routing schemes. In Arnold L. Rosenberg, editor, *Proceedings of the Thirteenth Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA 2001, Heraklion, Crete Island, Greece, July 4-6, 2001*, pages 1–10. ACM, 2001.

[Wag37] K. Wagner. Über eine eigenschaft der ebenen komplexe. *Mathematische Annalen*, 114:570–590, 1937.

[Wik24a] Wikipedia contributors. Domain name system — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Domain_Name_System&oldid=1228283936`, 2024. [Online; accessed 14-June-2024].

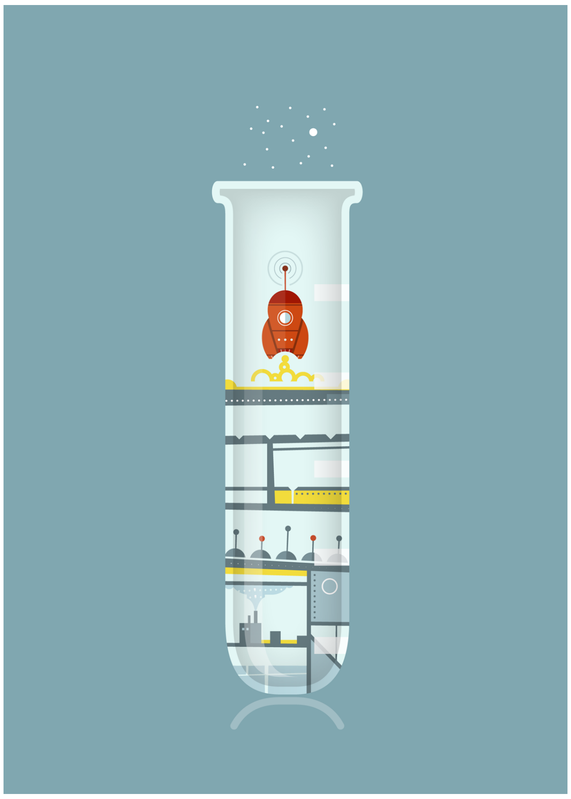
[Wik24b] Wikipedia contributors. Kazaa — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Kazaa&oldid=1210646824`, 2024. [Online; accessed 1-June-2024].

[Wik24c] Wikipedia contributors. Mastodon (social network) — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Mastodon_(social_network)&oldid=1225774368`, 2024. [Online; accessed 1-June-2024].

[Wik24d] Wikipedia contributors. Napster — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Napster&oldid=1225531804`, 2024. [Online; accessed 1-June-2024].

[Xu17] Shen Chen Xu. *Exponential Start Time Clustering and its Applications in Spectral Graphy Theory*. PhD thesis, Carnegie Mellon University, Pittsburgh, August 2017. CMU CS Tech Report CMU-CS-17-120.

[YJY$^+$15] Jiguo Yu, Lili Jia, Dongxiao Yu, Guangshun Li, and Xiuzhen Cheng. Minimum connected dominating set construction in wireless networks under the beeping model. In *2015 IEEE Conference on Computer Communications, INFOCOM 2015, Kowloon, Hong Kong, April 26 - May 1, 2015*, pages 972–980. IEEE, 2015.



This thesis was typeset using LaTeX, originally developed by Leslie Lamport and based on Donald Knuth's TeX. The body text is set in 11 point Egenolff-Berner Garamond, a revival of Claude Garamont's humanist typeface. The above illustration, "Science Experiment 02", was created by Ben Schlitter and released under CC BY-NC-ND 3.0. A template that can be used to format a PhD thesis with this look and feel has been released under the permissive MIT (XII) license, and can be found online at github.com/suchow/Dissertate or from its author, Jordan Suchow, at suchow@post.harvard.edu. This template was adapted by Nikolay Harutyunyan. You can get the source code to this template on Overleaf or from him, at nikolay.harutyunyan [at] fau.de. However, Thorsten Götte hacked around in this particular version.