Dissertationsschrift
zur Erlangung des Grades eines
Doktors der Naturwissenschaften

Disturbed Diffusive Processes for Solving Partitioning Problems on Graphs

von
Henning Meyerhenke

Paderborn, 4. April 2008

Fakultät für Elektrotechnik, Informatik und Mathematik
Universität Paderborn

Zusammenfassung

Die grundlegende Thematik der vorliegenden Dissertation ist die Identifizierung dichter Regionen eines ungerichteten Graphen $G = (V, E)$. Eine dichte Region ist dabei eine Teilmenge der Knoten $V' \subset V$ mit vielen Kanten, die zwischen Knoten aus V' verlaufen, aber vergleichsweise wenigen Kanten zu Knoten in $V \backslash V'$. Die Bestimmung dieser Gebiete ist bei der Lösung des Graphpartitionierungsproblems (GPP) sowie verwandten Aufgaben der Clusteranalyse hilfreich. Das GPP besteht in der Erstellung einer Partitionierung von V in k gleich große Teilmengen (Partitionen, Cluster) derart, dass die Zahl der Kanten, die zwischen verschiedenen Clustern verlaufen, minimiert wird. Es gibt zahlreiche Anwendungen, die die Lösung dieser oder ähnlicher Fragestellungen benutzen. Beispiele sind unter anderem parallele numerische Simulationen, Netzwerkanalyse, Schaltkreisentwurf sowie Genanalyse in der Bioinformatik.

GPP und alle relevanten Formulierungen verwandter Partitionierungsprobleme sind $\mathcal{NP}$-schwer, so dass keine Polynomialzeit-Algorithmen für ihre optimale Lösung bekannt sind. Partitionierungsbibliotheken, die dem aktuellen Stand der Technik entsprechen, benutzen lokale Knotenaustauschheuristiken innerhalb eines mehrstufigen Verbesserungsprozesses. Sie erzielen damit gute Lösungen in sehr kurzer Zeit. Jedoch entsprechen diese Lösungen nicht in jedem Fall den Anforderungen der Benutzer. Dies betrifft zum einen die Wahl der Zielfunktion im Optimierungsprozess, zum anderen die Form der Partitionen. Außerdem sind die am häufigsten eingesetzten Partitionierungsheuristiken schwierig zu parallelisieren, da sie inhärent sequentielle Teile enthalten. Eine solche Parallelisierung ist aber notwendig für den effizienten Einsatz als Lastbalancierer in parallelen Anwendungen. Zur Clusteranalyse von Graphen, bei der die Partitionen bzw. Cluster nicht gleich groß sein müssen, gibt es kein Verfahren, das sowohl sehr effizient arbeitet, Ergebnisse von sehr hoher Qualität in vielen verschiedenen Anwendungen liefert, als auch theoretisch wohlverstanden ist.

Um diese Nachteile bestehender Methoden zu beseitigen, entwerfen und untersuchen wir in dieser Arbeit den gestörten Diffusionsprozess FOS/C. Er kann dichte Gebiete eines Graphen von solchen mit wenigen Kanten unterscheiden, was wir mit seiner Beziehung zu Random Walks erklären. Durch die Kombination von FOS/C mit Bubble – einem generischen Verfahren ähnlich zu Lloyds k-means-Algorithmus – erhalten wir den iterativen und inhärent parallelen Algorithmus Bubble-FOS/C zur (Re)Partitionierung und Clusteranalyse von Graphen. In unseren theoretischen Untersuchungen zu FOS/C und Bubble-FOS/C beleuchten wir den Bezug zu Random Walks und zur Pseudoinversen der Laplacematrix des Eingabegraphen. Die dabei erzielten Ergebnisse führen unter an-

derem zu einem verbesserten Lösungsprozess von FOS/C und zu einem Beweis, dass Bubble-FOS/C gegen ein lokales Optimum konvergiert, welches durch eine Potentialfunktion charakterisiert werden kann.

Da Bubble-FOS/C die Lösung vieler linearer Gleichungssysteme erfordert, konstruieren wir einen effizienten Löser auf Basis des algebraischen Mehrgitterverfahrens (AMG). Die Graphhierarchie, die von diesem Löser erstellt wird, benutzen wir gleichzeitig für den mehrstufigen Partitionierungsprozess, der lokale Verbesserungen mit Bubble-FOS/C durchführt. Obwohl unser AMG-Löser eine deutliche Beschleunigung im Vergleich zu vorherigen Implementierungen hervorruft, bleibt die Laufzeit von Bubble-FOS/C weiterhin sehr hoch. Daher kann die gute Lösungsqualität des Algorithmus, die in Experimenten zur Graphpartitionierung beobachtet werden kann, kaum in der Praxis genutzt werden. Weitere Möglichkeiten zur Beschleunigung werden diskutiert, aber sie sind entweder nicht immer erfolgreich oder erfordern eine sehr aufwändige Implementierung.

Deshalb entwickeln wir in einem nächsten Schritt eine sehr viel schnellere und einfachere Methode zur Verbesserung von Partitionierungen. Diese Methode basiert auf einem anderen gestörten Diffusionsverfahren, das nur begrenzte Bereiche des Graphen betrachtet und auch einen hohen Grad an Parallelität aufweist. Das neue Verfahren kombinieren wir mit Bubble-FOS/C zu einem neuen mehrstufigen heuristischen Algorithmus namens DibaP. Eine Besonderheit dieser Kombination ist, dass ihre Mehrstufen-Hierarchie durch zwei verschiedene Konstruktionsansätze erstellt wird. Verglichen mit Bubble-FOS/C, zeigt die neue Heuristik eine deutliche Beschleunigung und erhält gleichzeitig die positiven Eigenschaften des langsameren Algorithmus. Ausführliche Experimente zeigen ein extrem gutes Verhalten bei der Partitionierung von Graphen, die aus numerischen Simulationen stammen. DibaP erzeugt durchgängig bessere Ergebnisse als die sehr häufig eingesetzten Bibliotheken METIS und Jostle. Weiterhin haben wir mit unserem neuen Algorithmus eine große Zahl der besten bekannten Partitionierungen von sechs weit verbreiteten Benchmark-Graphen verbessert. Auch in den verwandten Problemen der Lastbalancierung durch Repartitionierung und der Clusteranalyse verbessert DibaP die Lösungsqualität des Stands der Technik in vielen Fällen.

Insofern besteht die vorliegende Arbeit aus praktischen und theoretischen Fortschritten für die (Re)Partitionierung und die Clusteranalyse von Graphen durch die Entwicklung neuer erfolgreicher heuristischer Algorithmen und die theoretische Analyse einiger wichtiger Eigenschaften dieser Algorithmen.

Abstract

The underlying theme of this thesis is the detection of dense regions of an undirected graph $G = (V, E)$. A dense region is a subset of the nodes $V' \subset V$ with many edges between nodes in V' and only few edges to nodes in $V \backslash V'$. The identification of these regions is helpful for solving the graph partitioning problem (GPP) and related clustering tasks. The GPP asks for a partition of V into k equally sized subdomains (clusters) such that the number of inter-cluster edges is minimized. Applications that involve problems related to GPP are numerous; they include parallel numerical simulations, network analysis, circuit design, and gene analysis in bioinformatics.

GPP and all relevant formulations of related partitioning problems are $\mathcal{NP}$-hard, so that no polynomial-time algorithms for their optimal solution are known. State-of-the-art graph partitioning libraries employ local node-exchanging heuristics within a multilevel framework and yield good solutions in very short time. However, the computed partitions do not necessarily meet the requirements of all users. This includes the choice of the appropriate objective function and the shape of the computed subdomains. Furthermore, due to their sequential nature, the most popular partitioning heuristics are difficult to parallelize, which is necessary for their efficient use as load balancers in parallel applications. For graph clustering problems, where the cluster sizes do not need to be balanced, there is no method which is both highly efficient, delivers high-quality results in many diverse applications, and is theoretically well understood.

To overcome these drawbacks, we introduce the disturbed diffusion scheme FOS/C. It is capable of distinguishing dense from sparse graph regions, which we explain by its relation to random walks. The combination of FOS/C with the k-means related framework Bubble yields the iterative and inherently parallel (re)partitioning/clustering algorithm Bubble-FOS/C. In our theoretical investigations on FOS/C and Bubble-FOS/C, we examine the random walk relation and its connection to the pseudoinverse of the input graph's Laplacian matrix. Amongst others, the derived results lead to an enhanced solution process of FOS/C and to a proof that Bubble-FOS/C converges to a local optimum which can be characterized by a potential function.

Since Bubble-FOS/C requires the solution of many linear systems, we construct an efficient algebraic multigrid solver, whose graph hierarchy is simultaneously used for a multilevel improvement process of the partitions. Despite the fact that our algebraic multigrid approach is significantly faster than previous implementations, the running time of Bubble-FOS/C is still very high. Thus, its very good solution quality experienced in graph partitioning experiments can hardly be exploited in practice. Further

acceleration approaches are discussed, but they are either not always successful or very complicated to implement.

That is why we develop in a next step a much faster and easier method for the improvement of partitions. This method is based on a different disturbed diffusive process, which is restricted to local areas of the graph and also contains a high degree of parallelism. By coupling this new technique with Bubble-FOS/C in a multilevel framework based on two different hierarchy construction methods, we obtain our new heuristic DibaP for (re)partitioning and clustering graphs. Compared to Bubble-FOS/C, DibaP shows a considerable acceleration, while retaining the positive properties of the slower algorithm. Extensive experiments with popular benchmark graphs show an extremely good behavior for partitioning graphs stemming from numerical simulations. DibaP computes consistently better results than the state-of-the-art libraries METIS and Jostle. Moreover, with our new algorithm, we have improved a large number of the best known partitions of six widely used benchmark graphs. In the related problems of load balancing by repartitioning and graph clustering, DibaP also improves the solution quality of state-of-the-art programs in many cases.

Insofar, our work consists of practical and theoretical advances concerning graph (re)partitioning and graph clustering, achieved by the development of new successful heuristic algorithms and the theoretical analysis of some important properties of these algorithms.

Contents

1. Introduction

A natural way to express relationships between different entities are graphs. Each entity is modeled by a node and nodes are connected by edges if and only if they are related. In many applications it is of interest to know which nodes can be grouped together to form highly connected (*dense*) subgraphs based on these relationships. The problem of determining these subgraphs – which are also called *clusters*, *parts*, or *subdomains* – is the underlying theme of this thesis. Our results can be used for *graph partitioning*, *load balancing by repartitioning*, and *graph clustering*. In the following we describe what these terms refer to and which specific applications profit from an efficient and high-quality determination of dense subgraphs.

1.1. Applications

Graph partitioning is a widely used technique in computer science, engineering, and related fields. The most common formulation of the graph partitioning problem for an undirected graph $G = (V, E)$ asks for a division (*partition*) of V into k pairwise disjoint subsets of size at most $\lceil |V|/k \rceil$ such that the *edge-cut*, i.e., the total number of edges having their incident nodes in different subdomains, is minimized. Its applications include circuit layout [Fidu 82], air-traffic control [Bich 07], and the analysis of dynamical systems [Dell 06], to name only a few. They all have in common that edges between different subdomains have a different impact than internal edges. This impact can be a lower speed, a higher cost, or other undesirable features.

We mainly consider the use of graph partitioning for balancing the computational load in numerical simulations. Such simulations typically analyze natural or scientific processes that can be expressed via partial differential equations (PDEs). To make these equations solvable, they are discretized within the simulation domain, e. g., by the finite element method (FEM). Such a discretization yields a mesh, which can be regarded as a graph with geometric (and possibly other) information. Application areas of such simulations are fluid dynamics, structural mechanics, nuclear physics, and many others [Fox 94].

The solutions of discretized PDEs are often computed by iterative numerical solvers. Due to the involved computations and high memory space requirements, these solvers have become classical applications for parallel computers with many processing nodes connected by some communication network. To utilize all processors in an efficient manner, the computational tasks, represented by the mesh elements, must be distributed

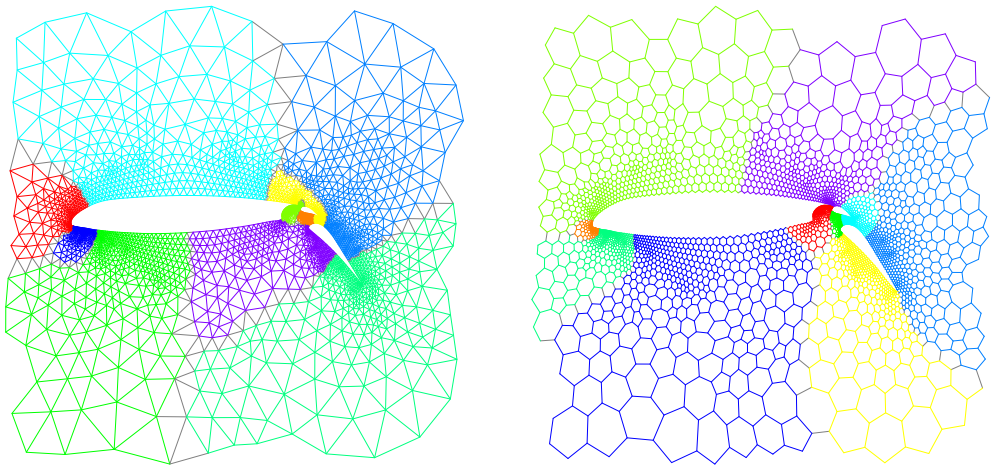


Figure 1.1.: Triangular mesh with holes representing airfoils (left) and its dual graph (right) with their respective partitions into ten subdomains. In each figure nodes of the same subdomain (and edges connecting them) have the same color; edges connecting nodes of different subdomains are shaded in grey.

onto the processors evenly. Moreover, the computational tasks of an iterative numerical solver depend on each other. Neighboring elements of the mesh need to exchange their values in every iteration to update their own value. Since inter-processor communication is much more expensive than local computation, neighboring mesh elements should reside on the same processor. Hence, a good distribution can be found by solving the graph partitioning (or a similar) problem for the mesh or its dual graph (in the dual graph G' of G, faces of G are replaced by nodes; these nodes in G' are connected by an edge if and only if their corresponding faces in G are adjacent, see Figure 1.1) [Schl 03].

Furthermore, in many numerical simulations some areas of the mesh are more interesting than others. For instance, during the simulation of the interaction of a gas bubble with a surrounding liquid, one is interested in the conditions very close to the boundary of the fluids rather than far away. To obtain an accurate solution, a high resolution of the mesh is required in the areas of interest. On the other hand, a uniformly high resolution is often not feasible due to limited main memory. That is why one has to work with different resolutions in different areas. Meshes with this property are called *adaptive*. The areas of interest may also change during the course of the simulation. In the above example, the gas bubble might change its position within the surrounding liquid, e. g., rise to the top. This behavior requires changes in the mesh, which may result in in load imbalances that delay the completion of the computation. Hence, after the mesh has been adapted again, its elements need to be redistributed such that every processor has a similar computational effort again. While this can be done by (re)partitioning the new mesh, the redistribution not only needs to find a new partition of high quality. It should also move as few nodes as possible to other processors because this *migration* causes high communication costs and (possibly expensive) changes in the local mesh data structure.

The task of clustering refers to the combination of objects to groups (clusters) such

that objects of the same group are more similar to each other than to objects from other groups [Jain 99], where similarity is usually application-dependent. It is a very important tool for the analysis and exploration of data arising in many different fields such as pattern recognition, bioinformatics, and business computing. In recent years the determination of clusters within graphs has received considerable attention, e. g., for image segmentation [Shi 00], detection of protein families [Enri 02], or the analysis of physical and social networks [Flak 02, Newm 04]. Generally speaking, clusters in a graph are dense node subsets that are only sparsely connected to each other. This notion is intentionally vague [Zhao 03], as it is again application-dependent and in general hard to formalize. In contrast to graph partitioning, the number of clusters is not always part of the input, and no explicit constraints on the cluster sizes are given.

1.2. Motivation

All relevant formulations of the aforementioned problems are combinatorially very hard to solve. More precisely, they are $\mathcal{NP}$-hard, so that no polynomial-time algorithms for their optimal solution are known. In practice fast heuristics are preferred, whose quality is usually determined experimentally.

State-of-the-art graph partitioning libraries employ node-exchanging heuristics within a multilevel framework for edge-cut minimization, cf. Section 1.3.1. They yield good solutions in very short time, but the computed partitions do not necessarily meet the requirements of all users. First of all, the edge-cut is not always a good measure for the total running time of parallel numerical simulations [Vand 95]. The number of *boundary vertices* (vertices that have a neighbor in a different subdomain), for instance, models the communication volume between processors more accurately than the edge-cut [Hend 98]. Moreover, the edge-cut is a summation norm, while often the maximum norm is of higher importance. For some applications, the *shape* of the subdomains plays a significant role. It can be assessed by various measures such as aspect ratio [Diek 00], maximum diameter [Pell 07a], connectedness, or smooth boundaries. Nevertheless, current partitioning-based load balancers do not take these facts fully into account.

While the total number of boundary vertices can be minimized by hypergraph partitioning [Devi 06], an optimization of partition shapes requires additional techniques (e. g., [Diek 00, Pell 07a]), which are far from being mature. Furthermore, due to their sequential nature, the most popular partitioning heuristics are difficult to parallelize. Although significant progress has been made (see Section 1.3.2), an inherently parallel graph partitioning algorithm can be expected to yield better solutions for partitioning as well as repartitioning, possibly also in shorter time.

The drawbacks of current graph clustering techniques are of a different nature. Due to the existence of various application-dependent graph clustering objectives and the hardness of their optimization, numerous algorithms and techniques for this problem have been developed, cf. Section 1.3.3. Unfortunately, as we will point out later in this

chapter, there is no method which is both efficient (i. e., with linear or close to linear running time), delivers high-quality results in many different application areas, and is theoretically well understood (e. g., with provable convergence in theoretical *and* practical settings). Closest to these requirements is the kernel k-means algorithm [Dhil 07]. Thus, its library implementation is used as a standard of comparison for our algorithms.

To overcome the drawbacks of the established heuristics, we follow a way of partitioning and clustering that is different from the techniques used in state-of-the-art libraries. It has been shown experimentally that a *shape-optimizing* approach (based on the BUBBLE framework or similar ideas) is very promising [Wals 95, Diek 00, Scha 05, Pell 07a], also see Section 1.3.4. The resulting graph partitions computed by most of these methods tend to be *convex* (curved outward), meet more requirements of the users than those of traditional node-exchanging heuristics, and induce small migration costs when used for load balancing by repartitioning. Moreover, the BUBBLE framework, whose idea is very similar to Lloyd's geometric k-means clustering algorithm [Lloy 82], is appealing in two additional respects. First, it uses a direct approach to obtain $k > 2$ subdomains, which is preferable to a recursive application of bisectioning (partitioning into two parts) [Simo 97]. The second advantage is that, in principle, the necessary operations for computing distances contain a high degree of parallelism. We circumvent previous problems of the BUBBLE framework by computing these distances or, more precisely, the similarities of nodes, by *disturbed diffusion schemes*, which do not require geometric information on the graph. The natural process diffusion (more details can be found in Section 2.2) sends load entities faster into densely connected regions of a graph than into sparse regions. As we will see later on, by disturbing the diffusive schemes, we avoid their balancing property and obtain efficient and effective mechanisms for distinguishing dense from sparse graph regions.

1.3. Related Work

1.3.1. Graph Partitioning

In order to structure previous related work on the graph partitioning problem, we use three different categories. The first category contains algorithms of mainly theoretical nature, which find approximate solutions to this $\mathcal{NP}$-hard problem. The second category contains heuristics that compute their solutions from scratch, having a global view on the input graph. In contrast to this, the heuristics in category three have only a local view, which means that they improve a given partition iteratively by changes in limited (local) areas of the graph.

1.3.1.1. Approximation Algorithms

Most approximation algorithms for graph partitioning or similar problems are either based on linear or semidefinite programming or on spectral arguments, i. e., properties

derived from the eigenvalues and -vectors of a matrix corresponding to the graph, cf. Khandekar et al. [Khan 06]. Up to now, the best approximation ratio is $\mathcal{O}(\sqrt{\log n})$ in time $\tilde{\mathcal{O}}(n^2)$ [Aror 04, Aror 07]. Alternatively, one can obtain subquadratic running time of $\tilde{\mathcal{O}}(n^{3/2})$ with an approximation ratio of $\mathcal{O}(\log n)$ [Aror 07]. Note that both running time values neglect polylogarithmic factors.

References to more results on approximation ratios, bounds on the solution quality (in particular for special graph classes), and approximation algorithms can be found in Monien et al. [Moni 06]. However, while these bounds and algorithms are of high theoretical importance, their practical relevance is in most cases minor. In practice a quadratic running time is prohibitive for large graphs, which are typical of numerical simulations or circuit layout. While having a worst-case deviation guarantee is certainly positive, practical heuristics might achieve better results on relevant real-world instances. In any case, both approximation algorithms use very involved algorithmic techniques, so that the $\mathcal{O}$-notation hides very large constants. That is why for applications with large inputs faster heuristics have been developed, which are described below.

1.3.1.2. Global Heuristics

Spectral methods. The fundamental idea of spectral partitioning methods dates back to Fiedler [Fied 73, Fied 75]. He has investigated what the second smallest eigenvalue λ_2 and its corresponding eigenvector z_2 (also known as *Fiedler vector*) of a graph's Laplacian matrix (cf. Chapter 2) reveal about the graph's structure. Based on these results, Pothen et al. [Poth 90] has developed a bipartitioning algorithm that decides the subdomain affiliation of a node depending on its entry in z_2. The reason for this is the connection to the integer program described in Section 2.1.1 that models the edge-cut minimization. If one relaxes the integer condition by allowing continuous solution values, the optimal relaxed solution is given by z_2. Let $\overline{m}$ be the median value in z_2. The bipartitioning can then be done as follows. All nodes with a larger value in z_2 than $\overline{m}$ are assigned to the first subdomain, all other nodes to the second one. This approach has been extended to more than two partitions without recursion by using more eigenvectors [Hend 95b]. The Lanczos algorithm (see e. g., [Demm 97]) makes the computation of the leading eigenvectors practical for large graphs, as each iteration has a complexity linear in the number of edges. Nevertheless, an accurate solution, which is reported to cost $\mathcal{O}(n^{3/2})$ total operations for sparse graphs in practice [Shi 00], is still more expensive than employing the subsequent geometric methods or local heuristics.

Geometric approaches and linear orderings. There exist numerous geometric partitioning methods which belong to the category of global heuristics. They require that the spatial location of each node is specified. One representative of such geometry-based algorithms are space-filling curves [Saga 94, Zumb 03]. They compute a linear ordering of the graph nodes, i. e., a bijective mapping from V to $\{1, \ldots, |V|\}$. This mapping aims at the preservation of the nodes' locality in space. Since this approach takes only

the geometric properties instead of the adjacency structure of the graph into account, the solution quality suffers if these two do not coincide. This happens for example in meshes that contain holes or fissures [Scha 04b]. Similar problems arise for other geometric approaches. Hence, despite their high speed and low memory consumption, the applicability of these methods is mostly limited to a restricted class of inputs. For this reason and since we do not require graphs to have geometric information in this thesis, we refer the interested reader to Schloegel et al. [Schl 03] for a more detailed description of geometric methods. More global heuristics, which are less important in our context, are described therein as well.

A method related to space-filling curves, which computes linear orderings without geometric information, uses so-called graph-filling curves (GFC) [Scha 04b]. The GFC solutions appear to be better than those of space-filling curves. Yet, they cannot compete with solutions computed by state-of-the-art multilevel algorithms described below.

1.3.1.3. Multilevel Paradigm

In most cases local heuristics can only be effective if they start with a reasonably good initial solution. Such a solution can be provided by the multilevel approach [Hend 95a], which has paved the way for nowadays successful local graph partitioning heuristics. The multilevel approach consists of three phases. Instead of computing a partition immediately for large input graphs, one computes a hierarchy of graphs $G_0, \ldots, G_l$ by recursive coarsening in the first phase. G_l ought to be very small in size, but similar in structure to the input graph G_0. Due to its small size, it is easy to compute a very good initial solution for G_l. This is done in the second phase, using one of different possible strategies such as spectral partitioning [Hend 95a] or coarsening until the number of remaining nodes equals the number of subdomains. After that, the solution is interpolated to the next-finer graph recursively. In this final phase the interpolated solution is refined using the desired local improvement algorithm, for example one of those described below.

The coarsening of the first phase ought to be very fast. It is typically done by computing a matching of the graph, which should have a high cardinality and a high weight. The matched nodes are combined to form super-nodes in the next hierarchy level. Different matching techniques have been used for this purpose, for example the two-approximation of a maximum weighted matching [Prei 99]. Being the first linear-time approximation algorithm for this problem, it has initiated further work, which has improved the approximation factor to $\frac{2}{3} - \epsilon$ [Drak 05]. A more detailed discussion of matching strategies can be found in Monien et al. [Moni 07] and an experimental comparison in Maue and Sanders [Maue 07]. Recently, and independently of our work, a coarsening based on algebraic multigrid techniques has been used in a multilevel algorithm for graph layout optimization [Safr 06].

The strength of the multilevel approach becomes already apparent if no local improvement in the second phase takes place. Even if the initial solution on the coarsest level is

determined randomly, this will usually yield a better solution than a random partition on the finest graph. This effect is due to the locality enhancement brought about by the coarsening process.

1.3.1.4. Local Heuristics

Kernighan-Lin and Helpful-Sets. Probably the most popular local heuristic for graph partitioning is the Kernighan-Lin (KL) heuristic [Kern 70], which has been developed originally for circuit partitioning. Its running time has been improved by Fiduccia and Mattheyses (FM) [Fidu 82] such that it is linear in the number of edges. As the main algorithmic idea has not changed, one often speaks of KL, although current implementations are based on the FM improvements. We refer to it either way.

Its idea is to improve an existing bipartition by performing node exchanges that reduce the edge-cut. In order to escape bad local optima, KL performs several passes, either a fixed number or until no further improvements can be found. In each pass it starts by computing for each node v how much the edge-cut would differ if v changed its subdomain. This difference value is called *gain*. Then, in an iterative process each node is moved logically exactly once to the other subdomain. The order in which this happens is based on the currently possible edge-cut gain yielded by the migration, from best to worst. After each migration of a node v, the respective gain value of v's neighbors are updated. Moreover, the situation is marked if its edge-cut value is the best so far and the subdomains are balanced. After all nodes have been examined, one determines if the best solution found in this pass is better than the one of the previous pass. If so, this local optimum is stored by performing all necessary moves physically.

Although no theoretical approximation rate guarantees are known for KL, its experimental results are convincing. If the initial partition is not extremely bad, this heuristic is able to find partitions with a good quality. Moreover, its fast running time makes it very appealing. That is why state-of-the-art partitioning libraries like METIS [Kary 98a] and JOSTLE [Wals 07a] and several others use some variant of the KL algorithm within a multilevel approach. Some of them use direct k-way implementations of KL. This is in principle much more complicated than the 2-way approach sketched above, but can be simplified to be practical [Kary 98b], resulting in a very effective algorithm. It should be mentioned, however, that the linear running time in $|E|$ is only possible if the edge weights take on discrete values of a finite interval. Only then the node order based on gain values can be generated this fast by a bucket-sort mechanism. A major drawback is the neglect of the resulting subdomain shapes, e. g., if they are connected or have small diameters. Moreover, the movement of nodes one after another is a strictly sequential process, which makes a parallelization very challenging. Successful attempts at such a parallelization are described in Section 1.3.2.

Similar to KL, the Helpful-Set (HS) heuristic for bipartitioning [Diek 95] uses local search based on node exchanges. It has evolved from a constructive proof by Hromkovič

and Monien on the bisection bandwidth of regular graphs [Hrom 91]. The main difference to KL is that HS migrates not only single nodes to the other subdomain, but also larger node sets. It is based on the concept of *helpfulness* of a node set. Analogous to the gain of a single node in KL, the helpfulness of a node set S is defined as the reduction of cut edges caused by migrating S. Starting with an initial balanced partition $\Pi = \pi_1 \cup \pi_2$, the HS algorithm reduces the cut size iteratively, until some termination criterion such as a desired cut size is met. In each iteration one searches for an s-helpful set in π_1, $s > 0$, and moves it to π_2. To restore the balance of Π, one searches for an s'-helpful set in π_2 of equal size with $s + s' > 0$ and moves it to π_1. In this way one ensures that the cut size is decreased in each iteration.

HS has been implemented in the partitioning library PARTY [Moni 00, Moni 04]. Its speed is nearly comparable to that of popular KL-based libraries, while the quality is reported to be often better in terms of the edge-cut [Scha 06]. We do not know of any successful parallelization of HS. As in KL, the node movement is a sequential process. It would be very difficult to ensure that different processors do not move the same node in different helpful sets at the same time.

Metaheuristics. In recent years a number of metaheuristics have been applied to graph partitioning (and graph clustering) problems. Some of these works use concepts that have already been very popular in other application domains such as genetic or evolutionary algorithms [Sope 04, Chev 06], multi-agent and ant-colony optimization [Koro 04, Come 06], and simulated annealing [Jerr 98]. Furthermore, two less established metaheuristics called PROBE (Population Reinforced Optimization Based Exploration) [Char 07] and Fusion Fission [Bich 07] have been adapted to or developed for graph partitioning. Most of these algorithms are able to produce solutions of a very high quality if they are allowed to run for a very long time. In practice, however, the running time investment necessary for good average solutions is too high, so that these methods are not widely used apart from special applications.

1.3.2. Load Balancing by Repartitioning

Recall that balancing the load in a numerical simulation requires the optimization of at least two objectives. On the one hand, it is crucial for an efficient simulation to compute a nearly balanced partition of high quality. On the other hand, the repartitioning process should not migrate too many vertices to different processors. Last but not least, load balancing should not be very expensive. Otherwise, it would not lead to running time savings within the whole simulation.

In order to consider these multiple objectives, different strategies have been explored in the literature. Two simple ones and their limitations are described by Schloegel et al. [Schl 97]. One is to compute a new graph partition from scratch and then to determine a migration-minimal mapping between the old and the new partition. This approach delivers good partitions. Yet, as the migration minimization is decoupled from

the partitioning process, the migration volume is often very high. Another strategy simply migrates vertices from overloaded subdomains to underloaded ones, until a new balanced partition is reached. While this leads to optimal migration costs, it often delivers partitions of poor quality. To improve these simple schemes, Schloegel et al. proposes a multilevel algorithm with three main features. First of all, the coarsening algorithm contracts only nodes of the same subdomain. By this means the coarsest partition still corresponds to the input partition. In the local improvement phase, two algorithms are used. On the coarse hierarchy levels, a diffusive scheme takes care of balancing the subdomain sizes. Since this might affect the partition quality negatively, a refinement algorithm is employed on the finer levels. It aims at edge-cut minimization by profitable swaps of boundary vertices. Subsequent work of the same authors combines the scratch-remap approach with the aforementioned diffusive methods to obtain the best of both schemes [Schl 00].

Diffusion has been used for load balancing by repartitioning as a means to compute how much load needs to be migrated between subdomains [Schl 01]. However, until recently, it has played only an implicit role in determining *which* elements should be migrated. Our methods and Pellegrini's algorithm ([Pell 07a] and Section 1.3.4.2) differ exactly in this respect from previous work, as their diffusive schemes direct the (re)partitioning process by computing the migrating elements explicitly.

Migration minimization with virtual vertices has been used by, amongst others, Hendrickson et al. [Hend 96]. For each subdomain an additional vertex is added, which may not change its affiliation. It is connected to each vertex v of this subdomain by an edge whose weight is proportional to the communication cost for moving v. In this way a partitioning of the new graph considers both migration costs and partition quality.

If one needs to balance the load in a parallel numerical application, one can assume that the application mesh is already stored in a distributed way on the processors. In this case, due to the resource limitations, it is often impossible, or at least not advisable, to repartition sequentially on one processor. Yet, while numerous high-quality sequential graph and hypergraph partitioning libraries exist (Chaco [Hend 94], Jostle [Wals 07a], METIS [Kary 98a], Party [Moni 04], PaToH [Cata 01], Plum [Olik 98], Scotch [Pell 07b], etc.), most of them for over a decade, the situation has been different for distributed/parallel repartitioning until recently. This is probably due to the increased complexity caused by parallel processing. The parallel versions of METIS [Kary 98a, Schl 02] and Jostle [Wals 97, Wals 00, Wals 07a] have been popular for several years, so that they will serve us as standard of reference. Only recently they have been supplemented by the load balancing toolkit Zoltan [Cata 07] with its parallel hypergraph partitioner (its solution quality for our benchmark graph – as opposed to hypergraph – problems is not competitive, however). All these parallel packages are mainly based on local improvement by KL/FM, but many also include techniques to circumvent some drawbacks of the latter.

1.3.3. Graph Clustering

The field of clustering has received considerable attention in the past decades. It is therefore impossible to give a complete overview in this thesis. Instead, we concentrate on graph clustering methods that are related to our techniques. This relation can be a similar approach or a similar objective. For more details on clustering methods not dealing with graphs, the interested reader is referred to Jain et al.'s survey [Jain 99]. An experimental study comparing different graph clustering algorithms and their clustering objectives has been performed by Brandes et al. [Bran 07].

One geometric clustering algorithm we do describe is Lloyd's k-means algorithm [Lloy 82]. It is one of the most popular algorithms for clustering geometric point data and related to our work despite its geometric nature. Lloyd's algorithm, whose input is a d-dimensional point set and the number of clusters k, starts by choosing for each cluster a representative center point. These centers do not have to be part of the input. The k-means objective function, which aims at the minimization of the sum of the squared Euclidean distances between each point p and the center of p's cluster, is $\mathcal{NP}$-hard to optimize globally for $k \geq 2$ [Drin 04]. Lloyd's algorithm finds a local optimum by two iterated alternating operations. The first one assigns each point to the cluster of its nearest center, while the second determines new centers for each cluster. The latter is done by choosing the respective center of gravity of a cluster. It can be shown that the objective function is locally minimized by these alternating operations, so that convergence is reached eventually [Seli 84].

1.3.3.1. Algorithms using Maximum Flow for Cut Minimization

A clustering method based on minimum cut trees has been suggested by Flake et al. [Flak 02]. Their algorithm uses the well-known relationship between maximum flows and minimum cuts (cf. e. g., [Corm 01]). It is shown that the density within some cluster and the sparsity of the edges between two clusters can be governed by a single parameter α. This α denotes the weight of additional edges, with which each node of the graph is connected to an artificial sink vertex. For this augmented graph a minimum cut tree is computed. After the removal of the artificial sink from this tree, its connected components are the clusters found by the algorithm. It should be noted that α controls indirectly the number of clusters, which is not predefined. By varying α, one can obtain a hierarchical clustering. For most practical cases the proposed method requires $\mathcal{O}(kn^{3/2})$ time, where k is the number of clusters found.

The basic idea of maximum flows is also utilized by Lang and Rao [Lang 04], whose algorithm MQI aims at the optimization of cut notions with quotients such as conductance or expansion for two clusters. Besides the graph, MQI takes an initial partition $\Pi = \pi_1 \dot{\cup} \pi_2$ as input and builds a directed flow network from it. Amongst other transformations, this is done by deleting all nodes of subdomain π_2 and connecting an artificial source to the nodes of π_1 that lie at the cut. All nodes of π_1 are also connected to an arti-

ficial sink by specifically weighted edges. The result of a maximum flow problem on this network yields an improved cut if it exists. It is shown experimentally that MQI delivers very good solutions in not much more than linear time by coupling METIS (for the initial solutions) with an efficient max-flow solver. Nevertheless, its applicability to general clustering problems is limited by the restriction to only two subdomains. A recursive application usually yields inferior results compared to direct k-way methods [Simo 97].

1.3.3.2. Graph Clustering with Random Walks

A random walk on a graph starts on a node v and then chooses the next node to visit from the set of neighbors (possibly including v itself) based on transition probabilities. The latter can for instance reflect the importance of an edge. This iterative process can be repeated an arbitrarily number of times. It is governed by the so-called *transition matrix*, whose entries denote the edges' transition probabilities. As a diffusion matrix is stochastic, it can be seen as such a transition matrix. More details about random walks (and their relations to diffusion) can be found in Lovász's survey [Lova 93].

Both diffusion and random walks are known to identify dense graph regions: Once a random walk reaches a dense region, it will stay there for a long time, before leaving it via one of the relatively few outgoing edges. An alternative view on this considers the adjacency matrix $\mathbf{A}$ of the graph and a corresponding transition matrix $\mathbf{P}$. An entry $\{u, v\}$ of the matrix $\mathbf{A}^t$ gives the number of paths from u to v with length t. Similarly, $\mathbf{P}^t_{\{u,v\}}$ denotes the probability of a random walk that starts in u to be located on v after t steps. If u and v are in the same dense region of the graph, this probability is above the average. One could also say that u and v are connected by many paths of short length.

The fact that random walks identify dense regions is used by van Dongen and his co-authors [Dong 00, Enri 02], who introduce a graph clustering algorithm that does not require the number of clusters a priori. It deals with the general problem arising whenever the t-step transition matrix $\mathbf{P}^t$ of a random walk is used to group graph nodes. If t is too large, $\mathbf{P}^t$ is close to the stationary distribution and contains hardly any information about the graph structure. On the other hand, t should be large enough to consider paths of a certain length. To bypass this dilemma, a nonlinear matrix operator which strengthens the differences between all rows of the matrix is combined with the traditional multiplication with $\mathbf{P}$. This leads to meaningful clusters, but the problem size is limited, as intermediate results include a densely populated matrix.

Similar concepts are used for the clustering algorithm by Harel and Koren [Hare 01]. The latter computes separator edges iteratively based on the similarity of their incident nodes. This similarity is derived from the sum of transition probabilities of random walks with very few steps. The iterative procedure is very fast and does not require the input of k. Unfortunately, it is not proven that it converges and it is also not made clear that the computed set of separator edges always leads to a reasonable clustering.

The strong connection between random walks and diffusion is also used by Lafon and

Lee [Lafo 06], whose key ideas are the definition of a diffusion-based distance measure between the nodes and a mapping of the nodes to a lower-dimensional space. This mapping yields an embedding that uses the principal eigenvectors of a diffusion kernel matrix as the basis of the image space. Thereby, the Euclidean distance of points in the image space corresponds (approximately) to their diffusion distance in the feature space. After the transformation one can apply established geometric clustering algorithms such as k-means, whose distance computations are then based on diffusion distances. The disadvantage of this approach is the expensive computation of eigenvectors. Moreover, the number of eigenvalues and -vectors needed for an accurate approximation is not known beforehand. It requires an intricate analysis of the matrix spectrum instead. This problem is actually related to the unknown random walk length t mentioned before.

A similar idea of defining a distance measure based on random walks and a corresponding embedding is followed by Fouss et al. [Fous 07], which utilizes the *commute time* of random walks as a distance measure. The commute time denotes for two nodes u, v the expected number of steps a random walk needs to start in u, visit v, and come back to u again. Due to the well-known relationship between electrical resistance and commute times [Doyl 84], it follows that this distance measure between nodes u and v decreases if the number of paths between these nodes increases. Fouss et al. shows that the commute time can be expressed as a distance measure based on the pseudoinverse (also known as *Moore-Penrose* inverse [Golu 96]) matrix of the graph's Laplacian. The resulting *Euclidean Commute Time Distance* (ECTD) follows the basic idea of Lafon's and Lee's diffusion distances. First of all, it determines how well-connected two nodes are. Secondly, it is shown that the feature space can be projected into a Euclidean subspace, where this projection approximately preserves the ECTD. Again, the embedded data can be clustered by established geometric clustering algorithms. However, while no eigenproblems have to be solved for this method, one of its drawbacks is that the computation of the Laplacian's pseudoinverse becomes "intractable" [Fous 07, p. 359] for large graphs. Although Fouss et al. describes an iterative procedure as a workaround to ease this problem, this does not change the fact that, in general, the pseudoinverse (even of sparse matrices) is a dense matrix whose computation has a lower bound of $\Omega(n^2)$.

1.3.3.3. Spectral Methods and Kernel k-means

In order to optimize quality measures based on cut notions, spectral methods have been used a number of times for graph clustering [Shi 00, Ng 01, Kann 04]. Analogous to their predecessors in graph partitioning, they are based on the relaxation of the integer program for the normalized cut, which is described in Section 2.1.3. The optimization of the more recent measure modularity has also been addressed by spectral techniques [Newm 06]. All such methods yield clusters with many intra-cluster edges and only few inter-cluster edges, but require costly eigenvector (or singular vector [Frit 08]) computations.

Zha et al. [Zha 01] and Ding et al. [Ding 04] relate k-means to principal component

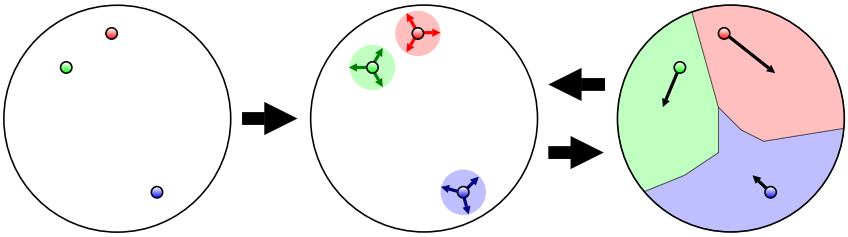


Figure 1.2.: Sketch of the main BUBBLE framework operations: Determine initial centers for each subdomain (left), assign each node to the subdomain of the nearest center (middle), and compute new subdomain centers (right).

analysis, which is a statistical method to describe data by a small number of important representatives. These representatives are in this case principal eigenvectors; all other data are given as their linear combinations. Based on these previous results, Dhillon et al. [Dhil 07] has developed the weighted kernel k-means (KKM) algorithm. Its authors show that KKM is a quite general and powerful approach, which can optimize a variety of graph partitioning and clustering objectives. Particularly appealing is that these objectives, which are frequently addressed by spectral methods, can be locally optimized by kernel k-means without expensive spectral algorithms. In each iteration of the algorithm, the objective is optimized by local changes of the cluster affiliation. This optimization is realized for each node by choosing the cluster with minimum distance, where this distance is computed in part by entries of a kernel matrix. KKM eliminates the problem that geometric k-means type algorithms can separate only convex clusters. By mapping the input to a higher-dimensional space using a nonlinear function, separating hyperplanes in this image space are nonlinear in the feature space.

A slight flaw of KKM is that its convergence is only guaranteed if a positive semidefinite kernel matrix is used. As the KKM authors point out, this convergence property should be sacrificed in favor of a better solution quality [Dhil 04, Dhil 07, Section 4.4], which can be obtained by using a negative diagonal shift of the kernel matrix. The larger this shift, the more are nodes willing to change their clusters, thereby reducing the likelihood of bad local optima. Kernel k-means has been implemented, together with some improvements like local search, in the graph clustering program GRACLUS [Dhil 07].

1.3.4. Bubble Framework and (Disturbed) Diffusion

The BUBBLE framework is related to Lloyd's k-means algorithm [Lloy 82] well-known in cluster analysis and transfers its ideas to graphs. Its first step is to choose initial cluster representatives (centers), one for each cluster. As illustrated in Figure 1.2, all remaining vertices are assigned to their closest center vertex w. r. t. some distance or similarity measure. After the subdomain assignment each cluster computes its new center for the next iteration. The two operations *assigning vertices to clusters* and *computing new centers* can be repeated alternately a fixed number of times or until a stable state is reached. For graph partitioning the algorithm has been introduced under the name

BUBBLE by Diekmann et al. [Diek 00], which provides references to previous related ideas like Walshaw et al. [Wals 95]. The name has been chosen because the assignment process resembles soap bubbles which grow simultaneously, starting at the centers and colliding at common borders. It is important to note that the actual implementation of the framework operations can differ significantly, as pointed out in the following.

1.3.4.1. Bubble Implementations based on Graph Distance or Geometry

A first implementation described (but not developed) by Schamberger [Scha 06, p. 66] relies on graph distances. It takes both a very long running time and delivers unsatisfactory results, so that we forgo a detailed description. A second approach is described by Diekmann et al. [Diek 00]. Here, the centers are distributed more evenly over the graph. This is accomplished by choosing only one initial center node at random. The others are chosen one after another furthest away from the current center nodes w. r. t. the graph distance. To compute the new subdomains, the smallest subdomain with at least one adjacent unassigned vertex grabs the vertex with the smallest Euclidean distance to its center. The new center of a partition is determined as the vertex for which the (approximate) sum of Euclidean distances to all other vertices of the same partition is minimal. These changes to the first approach solve some of its problems, for instance the initial center distribution is improved. Also, the computation of the new centers is sped up. Besides being connected, the subdomains are usually geometrically well-shaped by including coordinates in the choice of the next vertex. As a downside, the dependence on coordinates makes this version only applicable if such information is provided. Moreover, the Euclidean distance of two nodes might not coincide with the graph structure at all, leading to unsatisfactory solutions, as already explained for other geometric methods. Note that both of these BUBBLE implementations cannot be parallelized easily due to the strictly serial assignment process.

1.3.4.2. Disturbed Diffusion Schemes for Partitioning

In order to overcome the problems of previous BUBBLE implementations, Schamberger [Scha 04a, Scha 05] has developed two disturbed diffusion schemes called FOS/L and FOS/A. We call a diffusive mechanism *disturbed* if it is modified such that it does not result in a balanced load distribution, in which every node has the same amount of load. Integrated into BUBBLE, a mechanism based on disturbed diffusion is to reflect how well-connected the center nodes are to all other vertices of the graph. FOS/L achieves this by iterating FOS a fixed number of times, starting with a suitable initial load vector. In FOS/A one disturbs the iteration by a drain concept, where a small amount of load is shifted in each iteration to a set of source nodes. In a similar way the drain concept will be used for our scheme FOS/C. Thus, it is explained in more detail in the next chapter.

Schamberger's experiments show a promising partitioning quality of his methods. After several BUBBLE iterations the centers tend indeed to be within dense regions, while the

subdomain boundaries are often in sparse ones. However, he also points out that the practical relevance of his methods is very limited. Since an automatic procedure for determining a suitable number of iterations for FOS/L has not been found, this approach is very unreliable and needs extensive manual fine-tuning. The major drawback of FOS/A is its high running time because its convergence on large graphs is extremely slow. A theoretical problem is the changing amount of drain in FOS/A, depending on how much load a node has. This makes an analysis of the algorithm very difficult. It is therefore our objective to improve this situation by a faster and more reliable disturbed diffusion scheme, which is also easier to analyze.

Very recently, Pellegrini [Pell 07a] has addressed some drawbacks of the KL/FM heuristic. His partitioning approach aims at improved partition shapes, based on a diffusive mechanism used together with FM improvement. For the diffusion process the algorithm replaces whole partition regions not close to partition boundaries by one super-node. This replacement reduces the number of diffusive operations. As additionally the diffusion process is stopped when no more changes in the subdomain affiliation are expected, an acceptable overall speed is achieved. The implementation described is only capable of recursive bisection. As Pellegrini points out, a "full k-way diffusion algorithm is therefore required" [Pell 07a, p. 202] to improve the quality for large k.

1.4. Outline of our Results

The contribution of this thesis consists of both theoretical and practical results advancing the current state of graph partitioning, load balancing by repartitioning, and graph clustering. In summary these are the following:

- We introduce a new disturbed diffusion scheme called FOS/C and prove that it is a similarity measure. It does not require the specification of the number of diffusion iterations, which is accomplished by taking the limit of an infinite series. We prove that this infinite series and therefore FOS/C converges. Its convergence state can be computed by fast linear solvers, which is a major acceleration compared to the previous scheme FOS/A. To circumvent numerical issues, FOS/C is slightly altered by introducing a virtual vertex. This modification makes the solution process of the linear solvers faster and more robust. (Sections 3.1, 3.2, and 3.5)

- By relating FOS/C to random walks, we demonstrate why it is able to distinguish sparse from dense graph regions. Moreover, we prove that FOS/C computes entries of the pseudoinverse of the graph's Laplacian matrix, which plays a major role in the related Euclidean commute time distance (ECTD) measure for graph clustering. For distance-transitive graphs like the hypercube we show that the FOS/C convergence state (the Laplacian's pseudoinverse, respectively) can be characterized by means of a certain flow distribution. This characterization is shown not to hold in general for torus graphs. (Sections 3.2, 3.3 and 3.4)

- The integration of the similarity measure FOS/C into the Bubble framework yields the algorithm Bubble-FOS/C. We analyze the algorithm's complexity and make it applicable to graph clustering as well as graph (re)partitioning problems. Moreover, we give an indication why Bubble-FOS/C obtains the solution with the globally shortest boundary on the torus in our experiments. Our main theoretical result regarding Bubble-FOS/C is a proof based on a potential function, which shows that the algorithm always converges to a local optimum. To the best of our knowledge, this convergence proof and its potential function are the first substantial theoretical results on shape-optimizing graph partitioning algorithms and their solutions. We also prove that Bubble-FOS/C yields connected subdomains on vertex-transitive graphs when $k = 2$. (Sections 4.1, 4.2, 4.3, 4.5, and 4.7.1)

- Algebraic multigrid (AMG) is a fast solver for certain linear systems and has not been designed originally for semidefinite Laplacian system matrices, which arise in Bubble-FOS/C. After proving that AMG can be applied in principle to our problem class as well, we assemble and implement AMG components that suit our needs. Especially notable is the use of the AMG hierarchy not only for solving linear systems, but also for multilevel improvement in the partitioning/clustering process. As verified experimentally, our new AMG approach is able to speed up Bubble-FOS/C nearly five times compared to a related implementation that uses conjugate gradient as sparse iterative linear solver. (Sections 4.4, 4.6, and 4.7)

- Our experiments on FEM meshes also show that Bubble-FOS/C's implicit optimization of the subdomain shapes results in partitions of these graphs with subdomains that have short boundaries, good edge-cut values, low diameters, and are very often connected. Nevertheless, even with the acceleration by AMG, Bubble-FOS/C is up to three orders of magnitude slower than established graph partitioning libraries. Accelerations based on computing FOS/C on graph approximations instead of the whole graph are only partially helpful. (Sections 4.7 and 4.8).

- The fact that Bubble-FOS/C delivers high-quality graph partitioning solutions and that it has been studied theoretically, makes it very appealing as (re)partitioning tool. However, Bubble-FOS/C's high running time makes an exploitation of its solution quality hardly possible for large graphs occurring in practice. That is why we aim subsequently at the development of a faster heuristic that retains the positive properties of Bubble-FOS/C, but is significantly faster and suitable for practical deployment. As detailed below, our work in Chapter 5 achieves this objective and constitutes the most important part of this thesis from a practical point of view.

- Since we attribute the speed problem of Bubble-FOS/C to its global approach, we develop a faster diffusion-based method called TruncCons, which improves a given partition by local changes. We combine Bubble-FOS/C and Trunc-

CONS to obtain a linear-time (in $k \cdot |E|$) multilevel algorithm called DIBAP, which constitutes our main algorithmic achievement. The fine multilevel hierarchy levels are processed with fast algorithms for hierarchy construction (matchings) and local partition improvement (TRUNCCONS). Only on the coarse levels, BUBBLE-FOS/C is used to compute a good starting solution. (Sections 5.1, 5.2, and 5.3)

- The solution quality of DIBAP is excellent. In our experiments DIBAP delivers better graph partitioning solutions than the state-of-the-art partitioning libraries METIS and JOSTLE in terms of the edge-cut *and* the number of boundary vertices, both in the summation *and* in the maximum norm. Also problems from the two other considered fields repartitioning and clustering are nearly always solved with a comparable or better quality than by state-of-the-art libraries. Although DIBAP is still slower than established libraries for our three application domains, its running time is reasonable. (Section 5.4)

- Also notable is the fact that DIBAP improves for six benchmark graphs a large number (more than 80 out of 144) of their best known partitions w. r. t. the edge-cut. These six graphs are among the eight largest in a popular benchmark set [Sope 04, Wals 07b], which contains 34 graphs in total. (Section 5.4.1.6)

1.5. Publications

Parts of this thesis have been published in preliminary form in the proceedings of the subsequent peer-reviewed computer science conferences (followed by the reference of our respective contribution): 11th International Euro-Par Conference [Meye 05], 20th International Parallel and Distributed Processing Symposium [Meye 06a], 12th International Euro-Par Conference [Meye 06c], and 17th International Symposium on Algorithms and Computation [Meye 06b]. Our publication appearing in the proceedings of the 22nd International Parallel and Distributed Processing Symposium [Meye 08] has been selected by the program committee as the best paper of the conference's algorithm track.

Additionally, parts of this work have been presented at two events without refereed proceedings, the Dagstuhl Seminar *Web Information Retrieval and Linear Algebra Algorithms* (2007) and the *Oberwolfach Workshop on Algorithm Engineering* [Meye 07].

Note that many results of Chapters 3 and 4 in this thesis have been developed jointly with the co-authors of the aforementioned publications. In the following I will present algorithms and proofs which have been developed by myself or in collaboration. Proofs to which I have not contributed are omitted and replaced by a literature reference at the beginning of the result.

2. Preliminaries

In this section we define some terminology and give formal definitions of the partitioning problems considered in this thesis.

Definition 2.1. An edge-weighted graph $G = (V, E, \omega)$ is a triple with the set of *vertices* (or *nodes*) V, a set of *edges* $E \subseteq V \times V$, and an *edge weight function* $\omega : V \times V \to \mathbb{R}_{\geq 0}$. By definition, if $e \notin E$, we have $\omega(e) = 0$. If G is unweighted, we assume $\omega(u, v) = 1$ for all $(u, v) \in E$. G is called undirected if ω is symmetric, i. e., $\omega(u, v) = \omega(v, u)$ for all $(u, v) \in E$. An undirected edge between nodes u and v is written as $\{u, v\}$. Note that we sometimes write ω_e instead of $\omega(e)$. Usually, the number of nodes $|V|$ is denoted by n, the number of edges $|E|$ by m.

Remark 2.2. Note that, unless stated otherwise, we assume throughout this thesis that all graphs are undirected. While it is certainly possible to ask for partitions or clusterings of directed graphs, the majority of applications require only undirected graphs. It would be interesting to investigate in future work, however, if our methods can be extended to work for directed graphs, too. Moreover, we assume all graphs to be sparse, i. e., $m = \mathcal{O}(n)$. For most applications in our problem areas, sparseness is a reasonable or even natural assumption.

Remark 2.3. We also assume that the graphs to be partitioned or clustered are connected and simple, i. e., they do not contain self-loops (u, u) or multiple edges with the same endpoints. Additionally, we assume them to be finite. Finiteness and the lack of self-loops are natural assumptions, and connectedness can be simply enforced by focusing on the connected components. Furthermore, a graph with parallel edges can be transformed into an (equivalent) simple graph by merging multiple edges and adjusting the edge weight.

Notation 2.4. Matrices are written in bold font, but a matrix entry $[\mathbf{L}]_{u,v}$ is also written as $l_{u,v}$. Given a vector w, $[w]_v$ denotes its v-th component. Sometimes we also use w_v as a shorter variant of this notation. In case we refer to the v-th entry of the i-th vector in a series, we write $[w_i]_v$. The scalar (inner) product of two vectors x and y of length n is written as $\langle x, y \rangle = \sum_{i=1}^{n} x_i y_i$. Moreover, x and y are called *perpendicular*, denoted by $x \perp y$, if $\langle x, y \rangle = 0$.

Definition 2.5. The *Laplacian matrix* $\mathbf{L}$ of a graph G is defined as follows:

$$[\mathbf{L}]_{u,v} := \begin{cases} -\omega(e) & u \neq v, e = \{u, v\} \in E\,, \\ \deg(u) = -\sum_{q \neq u} [\mathbf{L}]_{u,q} & u = v\,, \\ 0 & \text{otherwise}\,. \end{cases}$$

Fact 2.6. *For undirected graphs,* $\mathbf{L}$ *is a symmetric positive semidefinite matrix [Chun 97]. Let the eigenvalues of* $\mathbf{L}$ *be denoted by* $\lambda_1 \leq \lambda_2 \leq \ldots \lambda_n$*. It is well-known that* $\lambda_1 = 0$ *and that the multiplicity of the eigenvalue* 0 *equals the number of connected components [Fied 73]. Hence, if* G *is connected,* $\lambda_2 > 0$ *and* $\mathbf{L}$ *has rank* $n-1$*, so that its null space* $\{x \in \mathbb{R}^n : \mathbf{L}x = \mathbf{0}\}$ *has dimension* 1*. The largest eigenvalue of a matrix is bounded from above by any induced matrix norm (e. g., [Meis 05]). Hence:* $\lambda_n \leq \|\mathbf{L}\|_1 = 2\,\mathrm{maxdeg}(G)$*, where* $\mathrm{maxdeg}(G) := \max\{[\mathbf{L}]_{u,u} \,|\, u \in V\}$ *denotes the maximum weighted degree of* G*.*

Note that in the following we use the terms *graph* and *matrix* interchangeably. The same holds for *edge weight* and *matrix* (off-diagonal) *entry*. If another matrix than the Laplacian is meant as graph representative, the meaning will be clear from the context.

2.1. Problem Definition

2.1.1. Graph Partitioning

Definition 2.7. Given an undirected graph $G = (V, E, \omega)$ with vertex set V of size n, edge set E of size m, and the edge weight function ω. Then, a *k-way partition* Π of G is a function

$$\Pi : V \to \{1, \ldots, k\}\,.$$

Such a partition divides the vertex set V into k disjoint subsets

$$V = \pi_1 \,\dot\cup\, \pi_2 \,\dot\cup \ldots \dot\cup\, \pi_k\,.$$

Edges connecting nodes of two different subdomains belong to the so-called *cut* of Π.

Definition 2.8. Let $G = (V, E, \omega)$ be a graph and let $\mathrm{dist}(u, v)$ denote the *graph distance* between nodes $u, v \in V$, i. e., the length of the shortest path connecting them. Moreover, let the affiliation of a node u to a subdomain π_c be either denoted as $\Pi(u) = c$ or as $u \in \pi_c$. Then, the quality measures *external edges* (or *cut edges*), *boundary nodes*, and *diameter* are defined for a subdomain π_c as

$$\begin{aligned}
ext(\pi_c) &:= \sum_{e \in \mathcal{C}} \omega(e) \text{ with } \mathcal{C} := \{e = \{u, v\} : \Pi(u) = c \wedge \Pi(v) \neq c\} \text{ (external edges)}\,,\\
bnd(\pi_c) &:= |\{v \in V : \Pi(v) = c \wedge \exists \{u, v\} \in E : \Pi(u) \neq c\}| \text{ (boundary nodes)}\,,\\
diam(\pi_c) &:= \max_{u,v \in \pi_c} \{\mathrm{dist}(u, v)\} \text{ (diameter)}\,.
\end{aligned}$$

If π_c forms more than one connected component in G, we call π_c *disconnected* and set $diam(\pi_c) := \infty$. Note that *bnd* can also be extended easily to node-weighted graphs by summing up the weights of boundary nodes instead of taking their number.

These three measures can be of different importance in different applications. External edges, for example, are used as an objective in circuit layout problems, where they

model connections between different modules. As these connections cause higher costs than wires within a module, they are undesirable [Kern 70]. In parallel applications, where the graph models data dependencies between objects, boundary nodes represent those objects which require inter-processor communication to obtain required data from neighboring nodes. Since inter-processor communication is much more expensive than local computation, the number of boundary nodes should be minimized [Hend 98]. Certain parallel numerical applications such as preconditioners additionally profit from good partition shapes. For a fast convergence of the underlying solvers, elongated subdomains with jagged boundaries should be avoided. Although the diameter does not measure such artifacts explicitly, it gives an indication if a subdomain is rather compact (i. e., resembles a circle) or elongated [Diek 00].

As the measures above are defined for one subdomain only, one needs to specify how the quality of the complete partition should be assessed. Again, this is application-dependent. For a chip design the total number of external edges is typically of highest importance. In contrast to this, parallel applications need to wait for the processor computing longest. There one should minimize the maximum number of boundary nodes. The sum and the maximum are the extreme cases ℓ_1 and ℓ_∞ of the more general ℓ_p-norms for a vector $x = (x_1, \ldots, x_n)^T$:

$$\|x\|_p \;=\; \left(\sum_{i=1}^{n} |x_i|^p\right)^{1/p} \quad \text{for } 1 \le p < \infty \text{ and } \|x\|_\infty = \max\{|x_i| : 1 \le i \le n\}\,.$$

Whereas ℓ_1 takes all entries of the input into account, ℓ_∞ focuses only on local behavior – the most extreme value. The norms in between can be used to measure both global and local appearance. In this thesis we consider only ℓ_1 and ℓ_∞ because none of the two should be neglected, as there are important applications for both of them. On the other hand, real applications for the other norms are less common.

The *edge-cut* of a partition Π, i. e., the weight of the edges whose endpoints belong to different parts, has been the most important graph partitioning metric. It is defined as

$$cut(\Pi) := \sum_{e=\{u,v\},\, \Pi(u)\neq\Pi(v)} \omega(e)\,,$$

which is half the summation norm ℓ_1 of ext. In case of an unweighted graph, it is just the number of edges running between different subdomains. The *balance* of Π is defined as

$$bal(\Pi) := \frac{\max_{1\le i\le k} |V_i|}{\lceil |V|/k \rceil}\,.$$

A partition is called *balanced* if $bal(\Pi) = 1$. The most common formulation of the graph partitioning problem is as follows:

Problem 2.9. Given an undirected graph $G = (V, E, \omega)$ and $k \in \mathbb{N}$, find a balanced k-way partition Π of G with minimum edge-cut.

This problem is known to be $\mathcal{NP}$-hard, and its decision variant is $\mathcal{NP}$-complete. This holds even if $k = 2$ (in which case Π is called a *bisection*) and all edge weights are one [Gare 74]. Hence, no deterministic polynomial-time algorithm is known to solve the graph partitioning problem optimally. Other variants of the traditional problem formulation of Definition 2.9 exist. For example, it is often the case that the balance constraint is relaxed to allow a small imbalance, e. g., $bal(\Pi) \leq 1.03$. It is frequently observed [Simo 97, Kary 98b] that this can lead to partitions with a higher quality. As indicated above, the maximum number of boundary nodes should be preferred in parallel numerical simulations for modeling the application's communication volume.

In spite of being only an approximation to the communication volume of the underlying numerical application, the edge-cut has been extremely popular as optimization objective in graph partitioning. Edge-cut minimization of a bipartition can be formulated as an integer program (e. g., [Lang 05]): Let $x_i \in \{-1, 1\}$ be an indicator variable, which denotes to which of the two subdomains node i belongs. Minimizing the edge-cut is then equivalent to minimizing the objective function $\frac{1}{4}x^T\mathbf{L}x$. $\mathbf{L}$ is the Laplacian matrix of the graph, and $\frac{1}{4}$ is included to model the number of cut edges exactly. The balance condition is expressed by the constraint $\sum_i x_i = 0$ in this quadratic integer program.

2.1.2. Load Balancing by Repartitioning

Load balancing is an essential tool in parallel processing for an efficient utilization of the computational resources. In this thesis we only consider load balancing of parallel computations in which the computations depend on each other. An example would be numerical simulations whose domains are discretized into meshes. These simulations typically employ iterative solvers which exchange information between neighboring mesh elements in every iteration. Inter-processor communication can be therefore minimized by a balanced partition with few boundary nodes.

If the mesh is severely altered during the simulation, intolerable load imbalances can arise. These are eliminated by computing a new balanced partition. A redistribution of the elements according to the new partition requires some nodes to change their processor. Such a change is called *migration*, which is defined as:

Definition 2.10. For a graph $G = (V, E)$, its old partition Π_1, and its new partition Π_2 the c-th entry ($1 \leq c \leq k$) of the vectors mig^{in} and mig^{out} is defined as

$$\begin{aligned} mig^{in}(c) &:= |\{v : \Pi_1(v) \neq c \wedge \Pi_2(v) = c\}| \text{ (incoming migration)}, \\ mig^{out}(c) &:= |\{v : \Pi_1(v) = c \wedge \Pi_2(v) \neq c\}| \text{ (outgoing migration)}. \end{aligned}$$

If we only speak of the migration volume mig without specifying if it is incoming or outgoing, we refer to either $mig_1 := \|mig^{in}\|_1 = \|mig^{out}\|_1$ for the summation norm or to $mig_\infty := \|mig^{in} + mig^{out}\|_\infty$ for the maximum norm.

Problem 2.11. Given an undirected graph $G = (V, E, \omega)$, $k \in \mathbb{N}$, and its k-way partition Π_1, find a new balanced k-way partition Π_2 of G such that

- the migration volume *mig* between Π_1 and Π_2 is minimized and
- Π_2 is optimized w. r. t. the edge-cut or the number of boundary nodes.

Which norm is chosen to measure the migration volume or the partition quality, depends on the application and should be chosen accordingly. Note that the quality of Π_2 can be measured in another metric such as the diameter, too.

Since both objectives (migration and partition quality) may contradict each other, a simultaneous optimization is often not possible. In these cases one can assign weights to the objectives and minimize their linear combination [Schl 00]. Then, the problem is obviously also $\mathcal{NP}$-hard since it solves the graph partitioning problem if the migration weight is set to zero. Alternatively, we can ask for a pareto-optimal solution, i. e., a solution for which there exists no other solution that is not worse in one objective and strictly better in the other one [Baño 06].

2.1.3. Graph Clustering

Recall that clustering refers in general to the placement of objects into groups (*clusters*) such that objects of the same group are similar to each other and objects of different groups are dissimilar. In graph terms this objective is translated into finding subsets of V that are densely connected within themselves, but sparsely connected to each other. Both of these formulations are very imprecise and underspecified. This is necessary because, as in graph partitioning, different applications require different objectives.

Mathematically, a clustering Π is also a partition of V, just as in Definition 2.7. The major difference between the graph partitioning problem and the graph clustering problem is that the latter requires (at least approximately) balanced partitions, while now cluster sizes can be (almost) arbitrary. Moreover, the number of subdomains k is known in the graph partitioning problem. For graph clustering, however, it can be, but does not have to be necessarily, part of the input. It is therefore often an advantage to employ an algorithm that does not require the specification of k. However, if k can be specified a priori, an exploitation of this information can be expected to improve the solution quality.

The multitude of applications for graph clustering has led to the existence of many different quality measures or objective functions. They can be used to steer the optimization process of the algorithm, to compare the results of different clustering algorithms, or to indicate whether the clustering found has too few or too many clusters. Most of them follow the paradigm of intra-cluster-density versus inter-cluster-sparsity [Gaer 05]. *Expansion* and *conductance* [Kann 04] are local measures taking the minimum of all subdomains. In contrast to them, *coverage* [Gaer 05], *modularity* [Newm 04], and *normalized cut [Shi 00]* are global measures taking the sum. The optimization of all these measures is

$\mathcal{NP}$-hard. For coverage this follows directly from the hardness of edge-cut minimization. Hardness proofs for the others can be found in the literature [Kaib 04, Sima 06, Shi 00]. The best approximation ratio for conductance known so far is $\mathcal{O}(\sqrt{\log n})$ [Aror 07].

In this thesis we focus on the normalized cut since it is a generalization of the edge-cut to imbalanced partitions. Moreover, it has a non-local view and has been applied successfully to a variety of applications [Dhil 07]. For a graph $G = (V, E, \omega)$ and its k-way clustering Π the normalized cut $NCut(\Pi)$ is defined as

$$NCut(\Pi) := \sum_{c=1}^{k} \frac{\sum_{u \in \pi_c, v \notin \pi_c} \omega(u, v)}{\sum_{u \in \pi_c, v \in V} \omega(u, v)} .$$

The generalization or normalization is necessary to cope with the missing constraint on the cluster sizes since an optimal k-clustering w. r. t. the edge-cut would simply cut off the $k-1$ nodes with smallest degree. Note that minimizing the normalized cut is equivalent to maximizing another measure, the normalized association [Shi 00]. The latter sums over the weight of the intra-cluster edges versus the weight of all edges in each cluster.

Just like in the case of the edge-cut, the minimization of the normalized cut for a bisection $\Pi = \{\pi_1, \pi_2\}$ of G can be expressed as an integer program [Shi 00]

$$\min_{\Pi} NCut(\Pi) = \min_{x} \frac{x^T \mathbf{L} x}{x^T \mathbf{D} x}$$

with the constraints $x_i \in \{1, -b\}$ and $x^T \mathbf{D} \cdot \mathbf{1} = 0$, where $b = \sum_{v \in \pi_1} \deg(v) / \sum_{u \in \pi_2} \deg(u)$ and $\mathbf{D}$ denotes the diagonal matrix of the node degrees in G. If the entries of x are allowed to take on real values, the relaxed optimal solution can be computed by solving the generalized eigenvalue problem $\mathbf{L}x = \lambda \mathbf{D}x$. This fact is exploited by spectral methods for graph clustering, which derive their cluster affiliation by interpreting one or more eigenvectors of $\mathbf{L}$.

Since our algorithms need the specification of the number of clusters k, we formulate the according graph clustering problem as follows:

Problem 2.12. Given an undirected graph $G = (V, E, \omega)$ and $k \in \mathbb{N}$, find a k-way clustering Π of G with minimum normalized cut.

The a priori specification of k can be seen as a limitation. A way of circumventing it, is to use a multilevel approach in which an algorithm not requiring the parameter k (e. g., the one of Enright et al. [Enri 02] or the one of Fritzsche et al. [Frit 08]), which computes the initial solution and k on a coarse representation of the input. Yet, such an approach is not pursued further here, but left to future work.

2.2. First Order Diffusion Scheme

Diffusive processes can be used to model a large variety of important transport phenomena. These phenomena arise in very diverse areas such as heat flow, particle motion

in solvents, and the spread of diseases. In a discrete setting on graphs, diffusion is an iterative process which exchanges splittable load entities between neighboring vertices, usually until all vertices have the same amount of load. That is why in computer science one has studied diffusion in graphs as one of the major tools for balancing the load in parallel computations [Xu 97].

The *general* or *first order diffusion scheme* (FOS) has been introduced independently by Cybenko [Cybe 89] and Boillat [Boil 90]. Since we often consider edge-weighted graphs (without node weights, unless stated otherwise explicitly), we use the extension to edge-weighted FOS by Diekmann et al. [Diek 99].

FOS belongs to the class of *local iterative algorithms* for balancing the load in independent parallel computations. Given a graph $G = (V, E, \omega)$ and a load (or *workload*) $w_v \in \mathbb{R}$ for each node $v \in V$, these algorithms distribute the total amount of load stepwise to the nodes of the graph. Finally, in the convergence state of these algorithms, each node has the same average amount of load. This process is performed by *local operations only*, i. e., only nodes adjacent to each other perform load exchanges. Below, we introduce FOS formally and present some results necessary to understand our work on disturbed diffusion.

Definition 2.13 (FOS)**.** Given a connected graph $G = (V, E, \omega)$, a suitably chosen constant $\alpha > 0$, and an initial load vector $w^{(0)} \in \mathbb{R}^n$. Let $w_u^{(t)}$ denote the load of node u in timestep t. Then, the edge-weighted first order diffusion scheme (FOS) performs the following operations in each iteration $t > 0$:

$$\begin{aligned} x_{e=\{u,v\}}^{(t-1)} &= \alpha\omega_e(w_u^{(t-1)} - w_v^{(t-1)}), \\ w_u^{(t)} &= w_u^{(t-1)} - \sum_{e=\{u,v\}\in E} x_e^{(t-1)}, \end{aligned}$$

where $x_{e=\{u,v\}}^{(t)}$ denotes the load exchange via edge e in iteration t.

In matrix-vector notation the FOS iteration of load updates can be written as $w^{(t)} = \mathbf{M}w^{(t-1)}$. The matrix $\mathbf{M} = \mathbf{I} - \alpha\mathbf{L}$ is the *diffusion matrix* of G. It is symmetric and doubly-stochastic, i. e., all entries are nonnegative and all row and column sums are one [Cybe 89].

The constant α is chosen such that the eigenvalues of $\mathbf{M}$, denoted by μ_i with $\mu_i = 1 - \alpha\lambda_i, 1 \le i \le n$, lie in the interval $(-1, 1]$. This can be achieved by $\alpha < \mathrm{maxdeg}(G)^{-1}$, where $\mathrm{maxdeg}(G) := \max_{1\le v\le n}\{\deg(v)\}$. A typical choice is $\alpha := (\mathrm{maxdeg}(G) + 1)^{-1}$. In case that G is bipartite, we additionally require that at least one diagonal entry of $\mathbf{M}$ is positive. Then, since $\mu_1 = 1$ and $|\mu_i| < 1$ for $2 \le i \le n$ if G is connected, the FOS iteration converges towards the *average* (or *balanced*) load situation $\overline{w} := \frac{1}{n}(\sum_{i=1}^n w_i^{(0)})(1, \ldots, 1)^T$, which has been shown by Cybenko [Cybe 89]. The convergence speed is dominated by $\gamma := \max\{|\mu_2|, |\mu_n|\}$, whose value depends on the graph structure.

Definition 2.14. Let $\mathbf{A} \in \{-1, 0, 1\}^{n \times m}$ be the node-edge incidence matrix of G (e. g., [Diek 99]) with $\mathbf{A}\mathbf{A}^T = \mathbf{L}$. Each column of $\mathbf{A}$ corresponds to an edge, each row to a node. Note that each column has exactly two nonzero entries, -1 and $+1$. In the column of edge $e = \{u, v\}$ the nonzero entries appear in the rows corresponding to the incident nodes u and v. In case of undirected graphs, the signs of the nonzero entries of $\mathbf{A}$ define an implicit (and arbitrary) direction of the edges. (In the following chapters we usually make the natural assumption that flow on an edge is directed from the node with higher load to the node with lower load.)

Definition 2.15. [Diek 99] A flow function $f : E \to \mathbb{R}$ is called *balancing* if and only if $\mathbf{A}f = w - \overline{w}$. The *FOS migrating flow* f^* is the sum of all load exchanges via the edges of G during the FOS iteration: $f^* := \sum_{t=0}^{\infty} x^{(t)}$.

Lemma 2.16. *[Hu 99, Diek 99] Let $\tilde{\mathbf{A}} = \mathbf{A}\mathbf{F}$ be the (edge-weighted) node-edge incidence matrix of $G = (V, E, \omega)$ and let $\mathbf{F}$ be an $m \times m$ diagonal matrix with $[\mathbf{F}]_{i,i} = \sqrt{\omega_i}$. The solution of the ℓ_2-minimization problem*

$$\textit{minimize } \|\mathbf{F}^{-1}f\|_2 \textit{ over all } f \textit{ with } \tilde{\mathbf{A}}f = d$$

is given by $f = \tilde{\mathbf{A}}^T z$, where $\mathbf{L}z = d$ with $d, z \in \mathbb{R}^n$, provided that $d \perp \overline{w}$. Using this minimization problem, it can be shown that the FOS migrating flow f^* is the unique $\|\cdot\|_2$-minimal balancing flow.

Note that in case one wants to minimize $\|\mathbf{F}f\|_2$ over all f with $\mathbf{A}\mathbf{F}^{-1}f = d$, one has to use the inverse $1/\omega_e$ instead of the edge weights ω_e in all diffusion formulas.

The most positive features of FOS are the locality of its operations and the optimality of the computed balancing flow. However, if used for load balancing, FOS should be replaced by the second order diffusion scheme SOS. The latter shares the same positive properties, but converges significantly faster towards the average load [Muth 98].

3. Disturbed Diffusion

A shape-optimizing approach to partitioning by means of the BUBBLE framework has been identified as very promising, see Schamberger [Scha 06] or Section 1.3.4 of this thesis. However, previous implementations of the BUBBLE operations are pre-mature and show some serious drawbacks, as pointed out in Section 1.3.4. In order to benefit from shape optimization, our objective is to overcome these drawbacks without requiring geometric information on the graph.

Regarding graph clustering, it is furthermore of interest to be able to group nodes based on their similarity. We therefore introduce in Section 3.1 a new disturbed diffusive process called FOS/C and prove some of its basic properties. It accomplishes significant advantages compared to the previous diffusion schemes in BUBBLE implementations, in particular w. r. t. robustness and computational requirements. As shown in Section 3.2 by its relation to random walks, FOS/C can be used as a similarity measure for the graph nodes. This measure regards two nodes or graph regions as similar if they are *well-connected*, which means that they are connected by many paths of short length. In Sections 3.3 and 3.4, the behavior of FOS/C on distance-transitive and torus graphs is investigated and several properties of the diffusive load distributions are derived. Finally, FOS/C is modified such that its solution can be computed faster. This acceleration is achieved in Section 3.5 by the introduction of a virtual vertex. The relation of this modified scheme to FOS/C and its relevant properties are also explored.

3.1. Disturbed Diffusion Scheme FOS/C

We call a diffusion scheme disturbed if it is modified such that it does not result in a balanced load distribution. In contrast to the ordinary first order diffusion scheme FOS, our new disturbed diffusion scheme FOS/C (C for constant drain) performs two load-changing operations in each iteration. While the first step is the original diffusion operation, the second one introduces a disturbance based on *drain*. It subtracts some fixed load amount δ (the drain) from each node and adds the total drain evenly onto some selected source nodes, denoted by the set $S \subset V$ (see Figure 3.1). This disturbance by the drain concept avoids the meaningless balanced load distribution in the convergence state, as we will see later on.

Definition 3.1. (FOS/C) Given a graph $G = (V, E, \omega)$, a set of source nodes $\emptyset \neq S \subset V$, and suitably chosen constants $\alpha > 0$ (cf. Section 2.2) and $\delta > 0$. Let the initial load

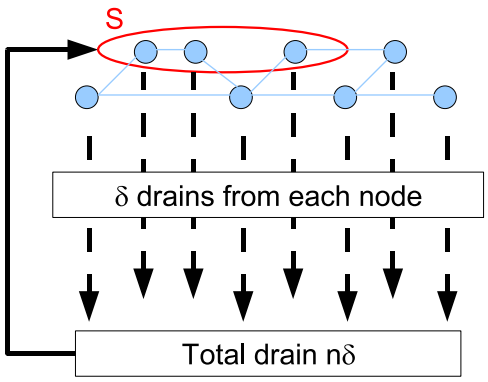


Figure 3.1.: Sketch of the drain concept with three nodes in the source set S.

vector $w^{(0)}$ and the drain vector d be defined as follows:

$$w_v^{(0)} = \begin{cases} \frac{n}{|S|} & v \in S\,, \\ 0 & \text{otherwise}\,, \end{cases} \qquad \text{and} \qquad d_v = \begin{cases} \frac{\delta n}{|S|} - \delta & v \in S\,, \\ -\delta & \text{otherwise}\,. \end{cases}$$

Then, the edge-weighted FOS/C diffusion scheme performs the following operations in each iteration $t > 0$:

$$\begin{aligned} x_{e=\{u,v\}}^{(t-1)} &= \alpha\omega_e(w_u^{(t-1)} - w_v^{(t-1)})\,, \\ w_u^{(t)} &= \Big(w_u^{(t-1)} - \sum_{e=\{u,v\}} x_e^{(t-1)}\Big) + d_u\,. \end{aligned}$$

The update of the load vector can be written in matrix-vector notation as $w^{(t)} = \mathbf{M}w^{(t-1)} + d$, with $\mathbf{M}$ being the diffusion matrix of G. Note that, since $\langle d, \mathbf{1}\rangle = 0$, this iterative load update does not change the total amount of system load. Node weights – if desired – can be incorporated into FOS/C by weighting the drain vector entries proportionally.

It requires a deeper analysis to see if FOS/C suits our needs, i. e., that load values can be derived from it which reflect how well-connected nodes or regions of a graph are with each other. That is why we need to know if FOS/C reaches a *convergence state*, where $w^{(t)}$ does not change any more. (In Markov chain theory such a state is also known as *steady state* or *stationary distribution*.)

Lemma 3.2. *Let* $\mathbf{M}$ *be a diffusion matrix and let* d *be a vector such that* $d \perp \mathbf{1} = (1, \ldots, 1)^T$. *Then,* $\lim_{t\to\infty}(\sum_{i=0}^{t} \mathbf{M}^i)d = (\mathbf{I} - \mathbf{M})^{-1}d$.

Proof. Recall that $\mathbf{1}$ is an eigenvector to the simple eigenvalue 1 of $\mathbf{M}$. Since $d \perp \mathbf{1}$, i. e.,

$\sum_{j=1}^{n} d_j = 0$, it follows that $\lim_{t\to\infty} \mathbf{M}^{t+1} d = 0$. Hence,

$$\begin{aligned} & \lim_{t\to\infty} (\mathbf{I} - \mathbf{M})(\mathbf{I} + \mathbf{M} + \mathbf{M}^2 + \cdots + \mathbf{M}^t) d \\ = & \lim_{t\to\infty} (\mathbf{I} - \mathbf{M}^{t+1}) d = \lim_{t\to\infty} d - \mathbf{M}^{t+1} d \\ = & \; d \,. \end{aligned}$$

Therefore, $(\mathbf{I} + \mathbf{M} + \mathbf{M}^2 + \cdots + \mathbf{M}^t)$ is the inverse to $(\mathbf{I} - \mathbf{M})$ in $(\mathbf{I} - \mathbf{M})d$ for $t \to \infty$ and any vector d perpendicular to $\mathbf{1}$, so that the claim follows. □

Theorem 3.3. *The FOS/C scheme converges for any arbitrary initial load vector $w^{(0)}$, provided that $d \perp \mathbf{1}$.*

Proof. Repeatedly applying the FOS/C update rule to the initial load vector $w^{(0)}$, we obtain

$$\begin{aligned} w^{(1)} &= \mathbf{M} w^{(0)} + d \\ w^{(2)} &= \mathbf{M} w^{(1)} + d = \mathbf{M}(\mathbf{M} w^{(0)} + d) + d = \mathbf{M}^2 w^{(0)} + (\mathbf{M} + \mathbf{I}) d \\ &\vdots \\ w^{(t)} &= \mathbf{M}^t w^{(0)} + (\sum_{i=0}^{t-1} \mathbf{M}^i) d \,. \end{aligned}$$

Due to the convergence of FOS to the average (balanced) load $\overline{w}$, Lemma 3.2, and $d \perp \mathbf{1}$ this yields

$$\begin{aligned} w^{(\infty)} &= \lim_{t\to\infty} \mathbf{M}^t w^{(0)} + (\mathbf{I} - \mathbf{M})^{-1} d \\ &= \overline{w} + (\alpha \mathbf{L})^{-1} d \,. \end{aligned}$$

□

Hence, the convergence state $w^{(\infty)}$ of FOS/C is composed of two parts. The first one is the balanced load distribution $\overline{w}$, while the second one depends only on α, $\mathbf{L}$, and d. Consequently, the first part is independent of the source set S, while the second one is independent of the initial load $w^{(0)}$. More precisely, the choice of $w^{(0)}$ only determines the total load within the system, i. e., the sum of all its entries. To make convergence loads of different source sets comparable, it is therefore necessary to fix this total load by using the same $\overline{w}$, e. g., the zero vector.

Note that, although the inverse $\mathbf{L}^{-1}$ itself does not exist, the last equation in the previous lemma is well-defined. The matrix $\mathbf{L}^{-1}$ acts directly on d, which is perpendicular to $\mathbf{1}$. The vector $\mathbf{1}$ is the eigenvector corresponding to the eigenvalue $\lambda_1 = 0$ of $\mathbf{L}$. Since $\mathbf{L}$ is a real symmetric matrix, its eigenvectors form a basis of $\mathbb{R}^n$ [Tref 97, Ch. 24]. Hence, we can represent d as a linear combination of the eigenvectors z_j of $\mathbf{L}$: $d = \sum_{j=1}^{n} a_j z_j$

with $a_j \in \mathbb{R}$. The property $d \perp \mathbf{1} = z_1$ can be written as

$$0 = \langle d, z_1 \rangle = \sum_{i=1}^{n} d_i \cdot [z_1]_i = \sum_{i=1}^{n} \sum_{j=1}^{n} a_j [z_j]_i \cdot [z_1]_i = \sum_{j=1}^{n} a_j \langle z_j, z_1 \rangle .$$

Since all the eigenvectors are orthogonal to each other, we have: $\sum_{j=1}^{n} a_j \langle z_j, z_1 \rangle = a_1 \langle z_1, z_1 \rangle$. As $\langle z_1, z_1 \rangle > 0$, the coefficient a_1 must be zero. This leads to the well-defined expression: $\mathbf{L}^{-1} d = \sum_{j=2}^{n} a_j \lambda_j^{-1} z_j$. Note that more details on the series $\lim_{t \to \infty} \sum_{i=0}^{t} \mathbf{M}^i d$ and the matrices involved are given in Section 3.2.

Corollary 3.4. *The convergence state $w^{(\infty)}$ of FOS/C exists and can be characterized as*

$$\begin{aligned} w^{(\infty)} &= \mathbf{M} w^{(\infty)} + d \\ \Leftrightarrow (\mathbf{I} - \mathbf{M}) w^{(\infty)} &= d \\ \Leftrightarrow \alpha \mathbf{L} w^{(\infty)} &= d . \end{aligned}$$

Thus, $w^{(\infty)}$ can be determined by solving the system of linear equations $\mathbf{L}w = d$, where $w = \alpha w^{(\infty)}$. We usually refer to the vector w as the convergence load vector.

Remark 3.5. If the set of source nodes S contains only one node, we call the computation of the FOS/C convergence state a *single-source FOS/C procedure*, otherwise it is called a *multiple-source FOS/C procedure*.

As we assume that G is sparse, w can be computed by sparse iterative methods with subquadratic space complexity. For example, a linear system $\mathbf{L}w = d$ describing the convergence state could be solved in principle by iterating FOS/C or similar diffusive methods. They have the advantage not to require global operations because nodes must exchange data only with their neighbors. Solvers such as Conjugate Gradient (CG) or multigrid methods [Saad 03] are preferable if global knowledge is available. They usually show a much faster convergence and a running time significantly below $\mathcal{O}(n^2)$. Our experiments on various benchmark graphs indicate that standard CG, using global operations, already yields a speedup of at least two orders of magnitude compared to the FOS/C iteration $w^{(t)} = \mathbf{M} w^{(t-1)} + d$.

Before we investigate the connection between FOS/C and random walks in the next section, we show some fundamental properties of FOS/C. This can be done by relating it to a flow problem and the $\|\cdot\|_2$-optimality of FOS.

Observation 3.6. *Since FOS/C solves $\mathbf{L}w = d$ for w with $d \perp \mathbf{1} = \overline{w}$, we can observe by using Lemma 2.16: The migrating flow $f = \mathbf{A}^T w$ in the FOS/C convergence state equals the $\|\cdot\|_2$-minimal flow that balances the vector d. In this load balancing problem, the nodes belonging to S send the respective load amount δ to all other nodes in the graph, which act as δ-consuming sinks.*

Corollary 3.7. *If either the flow or the loads in the convergence state are known, the respective other quantity can be computed easily:* $f_{e=\{u,v\}} = w_u - w_v$. *Note that the flow does not determine absolute values for the loads, only relative ones. To obtain absolute values from the flow, one needs to specify one of the load values.*

Proposition 3.8. *Consider the load vector* w *in the convergence state of FOS/C and the corresponding flow problem described in Observation 3.6. Then, the node* v *with maximum load value in* w *belongs to the set of source nodes* S.

Proof. Assume the opposite, i.e., $v \notin S$. Since v has the highest load, no flow is directed towards v because the flow on an edge is the load difference of its incident nodes. Hence, in the flow problem equivalent to FOS/C v does not receive any load. This is a contradiction to the initial setting because all nodes not in S receive a load amount of δ by definition. □

Proposition 3.9. *Let the graph* $G = (V, E, \omega)$ *and the load vector* w *of an FOS/C procedure with source set* S *be given. Then for each vertex* $v \in V$ *there is a path* $(v = v_0, v_1, \ldots, v_l = s)$ *with* $s \in S$ *and* $\{v_i, v_{i+1}\} \in E$ *such that* $w_{v_i} < w_{v_{i+1}}, 0 \leq i < l$.

Proof. Assume that the claim is untrue, so that no such monotonously increasing path exists. Moreover, recall that the convergence state of FOS/C is equivalent to a flow problem where all vertices $v \in V \backslash S$ receive a load amount of δ. Now, let j be the smallest index such that the monotonous path from v to $s \in S$ stops in $v_j \notin S$ because $w_{v_j} \geq w_{v'} \, \forall (v_j, v') \in E$. This means that v_j is a local maximum w. r. t. its load, so that the flow on its incident edges directs from v_j away. Hence, v_j would not receive any load. As all non-source vertices must receive a load amount of δ, our assumption is wrong and the claim true. □

3.2. Connections between FOS/C and Random Walks

Grouping nodes based on their similarity requires a formal notion of how similarity is determined and which important properties a similarity measure should have.

Definition 3.10. (comp. [Kauf 96, p. 440]) Let V be a finite set of nodes. We call a function $\mathcal{S} : V \times V \to \mathbb{R}$ a *similarity measure* for V if

- $\mathcal{S}(u, v) = \mathcal{S}(v, u)$ for all $u, v \in V$ and
- $\mathcal{S}(u, u) \geq \mathcal{S}(u, v)$ for all $u, v \in V$.

The symmetric matrix $\mathbf{S} = (s_{u,v} = \mathcal{S}(u, v))$ is called *similarity matrix*.

In order to show why FOS/C reveals a structural similarity between graph nodes or regions, we relate it to random walks.

Definition 3.11. A random walk on a graph $G = (V, E, \omega)$ is a discrete time stochastic process defined as follows: Starting on an initial node, a random walk performs the following in each iteration. First, it chooses one of the neighbors of the current node v randomly (where the probabilities are proportional to the edge weights). Then, it proceeds to the neighbor just chosen to start the next iteration. In some models one can also use a positive probability for staying on the current node.

As indicated before in Section 1.3, a random walk visiting a densely connected region of a graph is likely to visit many nodes of this region, before leaving it via one of the relatively few external edges. Moreover, the (shortest) paths between nodes of different clusters go all via these few external edges. On the other hand, nodes of the same cluster are connected to each other by many (shortest) paths of small length. This explains intuitively why random walks can be helpful for distinguishing internal from external edges or dense from sparse graph regions.

Based on how the transition probabilities are defined, different types of random walks can be distinguished. They have in common that a random walk has the *Markov property*. This means that the probability of going from node u to node v in timestep t depends only on the state in timestep $t-1$, not on the states in the timesteps before. Moreover, random walks are *time-homogeneous*, i. e., their transition probabilities are independent of the timestep.

It is well-known that ordinary, i. e., undisturbed, diffusion and random walks are closely related, see Lovász's survey on random walks [Lova 93]. In particular, the doubly-stochastic diffusion matrix $\mathbf{M}$ can be considered as the transition matrix of a random walk on $V(G)$. Using the random walk notion, $[\mathbf{M}]_{u,v}$ denotes the probability for a random walk located in node u to move to node v in the next timestep. In order to examine the relationship between *disturbed* diffusion and random walks, we show that the most important part of an FOS/C convergence load is the sum of random walk transition probabilities. These probabilities are determined by the diffusion matrix $\mathbf{M}$, and the random walks have an increasing number of steps.

Notation 3.12. Let $[w^{(t)}]_v^u$ ($[w^{(t)}]_v^S$) denote the load on node v in timestep t in a single-source (multiple-source) FOS/C procedure with node u as source (source set S). Recall that, whenever the timestep t is omitted, we refer to the convergence state of FOS/C.

Lemma 3.13. *Consider a multiple-source FOS/C procedure on graph $G = (V, E, \omega)$ with source set S. Then, for any node $v \in V$:*

$$[w^{(t)}]_v^S = [\mathbf{M}^t w^{(0)}]_v^S + \frac{n\delta}{|S|}\left(\sum_{i=0}^{t-1}\sum_{u\in S}[\mathbf{M}^i]_{v,u}\right) - t|S|\delta\,.$$

Proof. The FOS/C iteration scheme in timestep t for node v and source set S can be

written as (Theorem 3.3)

$$[w^{(t)}]_v^S \quad = \quad [\mathbf{M}^t w^{(0)} + \sum_{i=0}^{t-1} (\mathbf{M}^i) d]_v^S \,.$$

Now, we split the drain vector into two parts as $d = d_1 + d_2$, one for the source set S and one for the nodes in $V \backslash S$. The first part d_1 contains the entry $\frac{\delta n}{|S|} - \delta = \frac{\delta(n-|S|)}{|S|}$ in every row which corresponds to a node in S and zeros elsewhere. Similarly, d_2 has an entry of $-\delta$ in every row corresponding to a node in $V \backslash S$ and zeros elsewhere. This split, some rearranging, and using the fact that $\mathbf{M}$ is stochastic (has row sum 1) and a linear operator yield

$$\begin{aligned}
[w^{(t)}]_v^S &= [\mathbf{M}^t w^{(0)}]_v^S + [\sum_{i=0}^{t-1} \mathbf{M}^i d_1]_v^S + [\sum_{i=0}^{t-1} \mathbf{M}^i d_2]_v^S \\
&= [\mathbf{M}^t w^{(0)}]_v^S + \sum_{i=0}^{t-1} \left(\frac{\delta(n-|S|)}{|S|} \sum_{u \in S} [\mathbf{M}^i]_{v,u} + (-\delta) \sum_{u \notin S} [\mathbf{M}^i]_{v,u} \right) \\
&= [\mathbf{M}^t w^{(0)}]_v^S + \sum_{i=0}^{t-1} \left(\frac{\delta(n-|S|)}{|S|} \sum_{u \in S} [\mathbf{M}^i]_{v,u} - \delta \sum_{u \in S} (1 - [\mathbf{M}^i]_{v,u}) \right) \\
&= [\mathbf{M}^t w^{(0)}]_v^S + \frac{n\delta}{|S|} \left(\sum_{i=0}^{t-1} \sum_{u \in S} [\mathbf{M}^i]_{v,u} \right) - t|S|\delta \,.
\end{aligned}$$

□

Corollary 3.14. *Consider a single-source FOS/C procedure on graph $G = (V, E, \omega)$ with node $u \in V$ as source. Then, for any node $v \in V$:*

$$[w^{(t)}]_v^u = [\mathbf{M}^t w^{(0)}]_v^u + n\delta(\sum_{i=0}^{t-1} [\mathbf{M}^i]_{v,u}) - t\delta \,.$$

Proposition 3.15. *For any graph $G = (V, E)$ and two arbitrary, but fixed nodes $u, v \in V$ it holds:*

$$[w]_v^u = [w]_u^v \,.$$

Proof. Due to Corollary 3.14 we have

$$[w]_u^v - [w]_v^u \quad = \quad \lim_{t \to \infty} [\mathbf{M}^t w^{(0)}]_u^v - [\mathbf{M}^t w^{(0)}]_v^u + n\delta \left(\sum_{i=0}^{t} [\mathbf{M}^i]_{u,v} - \sum_{i=0}^{t} [\mathbf{M}^i]_{v,u} \right) - t\delta + t\delta \,.$$

The first two terms after the limit both converge towards the average load $\overline{w}$ [Cybe 89], also in the edge-weighted case [Diek 99]. Hence, they vanish just as the last two terms,

yielding

$$[w]_u^v - [w]_v^u = \lim_{t\to\infty} n\delta(\sum_{i=0}^{t}[\mathbf{M}^i]_{u,v} - [\mathbf{M}^i]_{v,u})\,.$$

As $\mathbf{M}$ is symmetric, all its powers are symmetric, too. Hence, all summands are zero, implying the claim. □

Considering the original FOS/C definition, the fact that this kind of load symmetry holds on all graphs is somewhat surprising. One might not expect such a property in graphs without any symmetry whatsoever. Its high relevance will become fully clear in Chapter 4, when the load symmetry is used to prove the convergence of the algorithm Bubble-FOS/C. Together with Proposition 3.8, we can deduce here that FOS/C is a similarity measure, which is important for its use in clustering algorithms:

Corollary 3.16. *As $[w]_v^u = [w]_u^v$ and $[w]_u^u > [w]_v^u$ for all $u, v \in V$, FOS/C is a similarity measure for the nodes of a graph. The matrix $\mathbf{W}' = (w'_{u,v} = [w]_v^u)$ is the FOS/C similarity matrix.*

Note that similarity measures are sometimes also required to lie in the interval $[0, 1]$, which is not fulfilled by FOS/C (but could be ensured by some suitable scaling).

Lemma 3.13 and the proof of Proposition 3.15 also show that for the interpretation of the load distribution in the convergence state only the sum term $n\delta(\sum_{i=0}^{\infty}[\mathbf{M}^i]_{u,v})$ in the middle part is of interest. The expression $[\mathbf{M}^i]_{u,v}$ denotes the probability of a random walk described by $\mathbf{M}$ to start in v and be located on u after i steps. In its spectral decomposition [Tref 97, Ch. 24], this matrix entry can be written as follows:

$$[\mathbf{M}^i]_{u,v} = \sum_{j=1}^{n} \mu_j^i [z_j]_u [z_j]_v\,,$$

where z_j denotes the j-th eigenvector and μ_j the j-th eigenvalue of $\mathbf{M}$. Recall that the largest absolute eigenvalue of $\mathbf{M}$ is $\mu_1 = 1$. It corresponds to the eigenvector $z_1 = (1, \ldots, 1)^T$ (or any scalar multiple of this vector). Since μ_1 is simple (Section 2.2), $|\mu_i| < 1$ for all $i > 1$. Hence, the μ_i^t with $2 \le i \le n$ converge to 0 if $t \to \infty$ and the limit of the spectral decomposition is

$$\lim_{t\to\infty} \sum_{j=1}^{n} \mu_j^t [z_j]_u [z_v]_v = [z_1]_u [z_1]_v\,.$$

Thus, all entries of $\mathbf{M}^t$ converge towards $[z_1]_u[z_1]_v$. As z_1 is the balanced distribution with all entries equal, the summands with large i in $\sum_{i=0}^{t-1}[\mathbf{M}^i]_{v,u}$ of Lemma 3.13 are of low importance. These values are already very similar, regardless of the choice of v and u. In contrast to this, the summands for small values of i reveal by the random walk interpretation if two nodes are connected to each other by many short paths or not.

One might wonder why it is necessary to iterate FOS/C for an infinite number of steps if only the first few iterates contribute significantly to the result. The reason is that by

taking the results of all random walks with lengths $0, \ldots, \infty$ into account, FOS/C can be used for general graphs without determining a *specific* suitable walk length. Hence, FOS/C is a very robust mechanism for identifying if two nodes u and v are densely connected to each other. This notion of connectedness can be extended to graph regions as well by using a larger source set S.

An alternative way of interpreting the convergence load of FOS/C uses random walk measures and their connection to a certain matrix. We explore this further connection in the following, which also yields an alternative proof for the FOS/C load symmetry.

Definition 3.17. Let $X_u^{(t)}$ be the random variable representing the node visited in timestep t by a random walk starting in u in timestep 0. Furthermore, let the balanced distribution vector be $\pi = (\frac{1}{n}, \ldots, \frac{1}{n})^T$ and let τ_u be defined as $\tau_u := \min\{t \geq 0 : X_u^{(t)} = s\}$ for any $u \in V$. Then, the (expected) *hitting time* H is defined as $H[u,s] := \mathbb{E}[\tau_u]$.[1] Moreover, the *commute time* $C[u,v]$ between nodes u and v is defined as $C[u,v] := H[u,v] + H[v,u]$.

One can describe the hitting time $H[u,v]$ as the expected timestep in which a random walk starting in u visits v for the first time. The commute time also includes the way back and is therefore symmetric. Fouss et al. [Fous 07] uses the square root of the commute time $C[u,v]$ as a distance measure between graph nodes $u, v \in V$. This Euclidean Commute Time Distance (ECTD) follows a similar idea as FOS/C of reflecting how well-connected two nodes are. The commute time can be computed by using the following lemma:

Lemma 3.18. *[Fous 07] The commute time between nodes u and v can be computed as $C[u,v] = vol_G(l^\dagger_{u,u} + l^\dagger_{v,v} - 2l^\dagger_{u,v})$, where vol_G is the volume of graph G, $vol_G = \sum_{j=1}^{n} \deg(j)$.*

The matrix $\mathbf{L}^\dagger$ is called *(Moore-Penrose) pseudoinverse* [Golu 96, p. 257f.] or *discrete Green's function* [Elli 01a] of $\mathbf{L}$. Like $\mathbf{L}$, it is symmetric positive semidefinite and doubly centered, i. e., both row sum and column sum are zero. If $(\lambda_i \neq 0, z_i)$ is the i-th pair of eigenvalues/-vectors of $\mathbf{L}$, (λ_i^{-1}, z_i) is the analogous i-th pair of $\mathbf{L}^\dagger$. All pairs $(\lambda_i = 0, z_i)$ are eigenvalues/-vectors of both $\mathbf{L}$ and $\mathbf{L}^\dagger$ (comp. [Fous 07]). Thus: $[\mathbf{L}^\dagger]_{u,v} = \sum_{i=2}^{n} \lambda_i^{-1} [z_i]_u [z_i]_v$. The pseudoinverse can also be used to compute the FOS/C convergence load vector w directly. One way to see this is to consider w in $\mathbf{L}w = d$ as a solution to the least square problem $\min_{w \in \mathbb{R}^n} \|d - \mathbf{L}w\|_2$. It is known that $w = \mathbf{L}^\dagger d$ provides the solution to this minimization problem [Golu 96, p. 256f.]. Since $d \perp \mathbf{1}$, a solution w which attains the minimum value 0 exists (Corollary 3.4), so that the FOS/C convergence vector w can also be stated as $w = \mathbf{L}^\dagger d$.

Consequently, apart from linear solvers, the convergence load $w = \alpha w^{(\infty)}$ can be computed by iterating FOS/C (i. e., by successive matrix-vector products $\mathbf{M}^t d$ similar to the

[1] Note that this definition, which is also used by other authors [Norr 97, Fous 07], yields $H[u,u] = 0$ for all $u \in V$. Yet, alternative formulations also exist, which result in $H[u,u] \neq 0$ for all $u \in V$ [Boll 98].

power iteration method for computing eigenvectors [Golu 96]), or by using the eigenvalues and -vectors of $\mathbf{L}^\dagger$, or by direct pseudoinversion. Yet, direct pseudoinversion is for general graphs as complex as inversion and requires $\Omega(n^2)$ operations [Elli 01a]. Also the former two approaches are not recommended in practice. Power iteration methods usually converge very slowly [Golu 96, Ch. 7.3], which we could confirm in our own experiments. Similarly, computing (nearly) all the eigenvalues and -vectors of $\mathbf{L}^\dagger$, even with a fast eigensolver, is computationally very expensive, too. Storing all these eigenvectors or the (in general dense) matrix $\mathbf{L}^\dagger$ also requires $\mathcal{O}(n^2)$ storage. This is only possible for rather small graphs. On the other hand, if $\mathbf{L}^\dagger$ is already known, FOS/C convergence loads can be computed very easily. In any case, the following results provide an elegant interpretation of these load values in terms of $\mathbf{L}^\dagger$.

Proposition 3.19. *Let S be the source set in a multiple-source FOS/C procedure with $\overline{w} = (0, \ldots, 0)^T$. Then, the entries of the FOS/C convergence load vector w can be computed as*

$$[w]_v^S = [\mathbf{L}^\dagger d]_v^S = \frac{n\delta}{|S|} \sum_{u \in S} l_{v,u}^\dagger \,.$$

Proof. Recall that the row sum of $\mathbf{L}^\dagger$ is always 0 and that $w = \mathbf{L}^\dagger d$. Hence, by splitting the drain vector into two parts, one with only negative entries, one with only positive ones, we obtain

$$\begin{aligned} [w]_v^S = [\mathbf{L}^\dagger d]_v^S = \sum_{u=1}^{n} l_{v,u}^\dagger d_u &= \frac{\delta(n-|S|)}{|S|} \sum_{u \in S} l_{v,u}^\dagger - \delta \sum_{u \notin S} l_{v,u}^\dagger \\ &= \frac{\delta(n-|S|)}{|S|} \sum_{u \in S} l_{v,u}^\dagger + \delta \sum_{u \in S} l_{v,u}^\dagger - \delta \sum_{u=1}^{n} l_{v,u}^\dagger \\ &= \frac{n\delta}{|S|} \sum_{u \in S} l_{v,u}^\dagger \,. \end{aligned}$$

□

Corollary 3.20. *Let s be the source in a single-source FOS/C procedure. Then, the entries of the FOS/C convergence load vector w can be computed as*

$$[w]_v^s = [\mathbf{L}^\dagger d]_v^s = n\delta \cdot l_{v,s}^\dagger \,.$$

Consequently, the FOS/C similarity matrix equals the pseudoinverse of the graph's Laplacian up to scaling. Now, the load symmetry of FOS/C also follows from the fact that $\mathbf{L}^\dagger$ is a symmetric matrix.

Returning to the random walk notion, both measures hitting and commute time can be related to FOS/C, too, by the following results.

Corollary 3.21. *By using the formulas of Lemma 3.18 and Corollary 3.20, the commute*

time between nodes u and v can be expressed in terms of FOS/C convergence loads as

$$C[u,v] = vol_G \left(\frac{[w]_u^u + [w]_v^v - 2[w]_v^u}{\delta n} \right) .$$

Theorem 3.22. *[Meye 06b] In the convergence state it holds for two nodes $u, v \in V$ not necessarily distinct from a source $s \in V$:*

$$[w]_u^s - [w]_v^s = \delta(H[v,s] - H[u,s]) .$$

The main commonality of ECTD and FOS/C is the underlying notion of random walks to determine the similarity of graph nodes. Both can express this notion by means of the pseudoinverse of the graph's Laplacian matrix. As we will show next, on vertex-transitive graphs (cf. Definition 3.25) a maximization of the FOS/C similarity is even equivalent to the minimization of the commute time distance. Hence, embedded into center-based clustering algorithms (i. e., where each cluster has one distinguished center node to which distances are computed) such as k-means, both measures yield the same clustering results on vertex-transitive graphs such as the hypercube or the torus (see Sections 3.3 and 3.4).

Theorem 3.23. *For arbitrary nodes $u, v \in V$ of a vertex-transitive (unweighted) graph $G = (V, E)$ and a source set $S \subset V$ it holds:*

$$\arg\min_{u \in V} \sum_{v \in S} C[u,v] = \arg\max_{u \in V} [w]_u^S .$$

Similarly, for a node $v \in V$ and nodes $u_1, \ldots, u_k \in V$:

$$\arg \min_{c=1,\ldots,k} C[u_c, v] = \arg \max_{c=1,\ldots,k} [w]_v^{u_c} .$$

Proof. Due to a result of Alon and Spencer (see Lemma 3.42) we have $[\mathbf{M}^t]_{v,v} = [\mathbf{M}^t]_{u,u}$ for all $u, v \in V$ of an unweighted vertex-transitive graph G. Combining Proposition 3.15 and Corollary 3.20, we get:

$$l^\dagger_{u,u} - l^\dagger_{v,v} = \frac{[w]_u^u - [w]_v^v}{n\delta} = \lim_{t \to \infty} \sum_{i=0}^{t} [\mathbf{M}^i]_{u,u} - [\mathbf{M}^i]_{v,v} = 0 .$$

Since all diagonal entries of $\mathbf{L}^\dagger(G)$ are equal, the values vol_G, $l^\dagger_{u,u}$, and $l^\dagger_{v,v}$ can be seen as constants, yielding

$$\begin{aligned} \arg\min_{u \in V} \sum_{v \in S} C[u,v] &= \arg\min_{u \in V} \sum_{v \in S} vol_G(l^\dagger_{u,u} + l^\dagger_{v,v} - 2l^\dagger_{u,v}) \\ &= \arg\min_{u \in V} \sum_{v \in S} -2l^\dagger_{u,v} = \arg\max_{u \in V} \sum_{v \in S} l^\dagger_{u,v} \\ &= \arg\max_{u \in V} [w]_u^S . \end{aligned}$$

A similar series of arguments yields: $\arg\min_{c=1,\ldots,k} C[u_c, v] = \arg\max_{c=1,\ldots,k} l^{\dagger}_{u_c,v} = \arg\max_{c=1,\ldots,k}[w]_v^{u_c}$. □

An important aspect where the methods differ from each other is crucial for their complexity and in favor of FOS/C. To compute the distance to some node v for all other nodes within a center-based clustering algorithm, FOS/C requires only the v-th row of $\mathbf{L}^{\dagger}$, which is computed by solving a single-source procedure. ECTD, however, also needs all diagonal entries of $\mathbf{L}^{\dagger}$. To determine them by computing the whole pseudoinverse matrix, becomes intractable for larger problems, as also remarked by Fouss et al. [Fous 07]. Both running time, which is at least quadratic, and space consumption become prohibitive. By using additional techniques such as a sparse Cholesky factorization, Fouss et al. has been able to tackle sparse graphs with up to 150,000 nodes. Compared to what is theoretically feasible with sparse iterative linear solvers for FOS/C on commodity hardware (sparse linear systems with a few million variables can be solved on any modern desktop computer with 2 GB main memory), the quantity 150,000 is relatively small.

3.3. FOS/C on Distance-Transitive Graphs

After these results on general graphs, we turn our attention to the behavior of FOS/C on two specific, but important, graph classes. In view of the previous findings, one can see this also as an analysis of the Laplacian's pseudoinverse whenever the convergence state of FOS/C is concerned. Note that, as pointed out by Ellis [Elli 01a], closed-form functions for the pseudoinverse must be computed for each new class of graphs.

This section deals with distance-transitive graphs, a class of which several representatives are very important in parallel network topologies and coding theory. We show that the FOS/C load distribution (respectively the entries of $\mathbf{L}^{\dagger}$) can be computed for this class by relating it to a flow problem. The graphs considered in this section are assumed to be unweighted and all FOS/C procedures have only one single source.

Definition 3.24. [Bigg 93, p. 115] Given a graph $G = (V, E)$, a permutation π of V is an *automorphism* of G if

$$\{u, v\} \in E \Leftrightarrow \{\pi(u), \pi(v)\} \in E, \forall u, v \in V\,.$$

The set of all automorphisms of G, with the operation of composition, is the automorphism group of G, denoted by $Aut(G)$.

Definition 3.25. [Gros 04, p. 12] A graph $G = (V, E)$ is *vertex-transitive* if for any two distinct vertices of V there is an automorphism mapping one to the other.

An even stronger property is distance-transitivity:

Definition 3.26. [Bigg 93, p. 118] A graph $G = (V, E)$ is *distance-transitive* if, for all vertices $u, v, x, y \in V$ such that $\mathrm{dist}(u, v) = \mathrm{dist}(x, y)$, there exists an automorphism φ for which $\varphi(u) = x$ and $\varphi(v) = y$.

Distance-transitive graphs are symmetric graphs and therefore vertex-transitive, edge-transitive, and regular [Bigg 93, Chs. 15 and 20]. One important subclass of distance-transitive graphs are Hamming graphs [Bon 07]. The concept of Hamming distance, represented by path lengths in Hamming graphs, frequently occurs in coding theory for error detection and correction [Adam 91]. A very well-known Hamming graph is the hypercube [Leig 92], which is also important as a topology for connecting parallel processors. Other distance-transitive graphs are the Petersen graph, complete graphs, and complete bipartite graphs with parts of equal size [Gods 01, Ch. 4.5].

Definition 3.27. Let $N_i(u) := \{v \in V | \operatorname{dist}(u,v) = i\}$ denote the i-neighborhood of u. A graph $G = (V, E)$ has a level structure (is a level structure graph) w. r. t. a node $s \in V$ if V can be partitioned into levels $\{s\} = L_0, L_1, \ldots, L_\Lambda$ such that for all $0 \leq i \leq \Lambda$:

$$\begin{aligned} \forall u, v \in L_i \; \forall j \in \{0, \ldots, \Lambda\} : |N_1(u) \cap L_j| &= |N_1(v) \cap L_j| \\ \text{and } L_0 \dot\cup \ldots \dot\cup L_\Lambda &= V\,. \end{aligned}$$

Lemma 3.28. *[Bigg 93, p. 155f.][Gods 01, p. 67] If $G = (V, E)$ is distance-transitive, then $N_i(s)$ forms the i-th level $L_i(s)$ of a level structure in G w. r. t. an arbitrary, but fixed node $s \in V$.*

As an example, the κ-dimensional hypercube $Q(\kappa)$ has $\Lambda = \kappa + 1$ such levels. The results of this section can be derived by means of this level structure and the equivalence of FOS/C to the following $\|\cdot\|_2$-minimal flow problem.

Definition 3.29. Consider the flow problem of Observation 3.6, where s sends a load amount of δ to all other vertices of G, which act as δ-consuming sinks. If the flow is distributed such that for all $v \in V \backslash \{s\}$ the same flow amount is routed on every (not necessarily edge-disjoint) shortest path from s to v, we call this the *uniform flow distribution*. Note that in case more than one shortest path traverses the same edge e, the total flow on e is the flow sum of all shortest paths via e.

The reason, why this flow problem is interesting, is its connection to FOS/C. Recall from Observation 3.6 that the $\|\cdot\|_2$-minimal solution of the flow problem of Definition 3.29 is equal to the solution of the underlying FOS/C procedure.

Proposition 3.30. *Let G be a distance-transitive graph. Then, $w_u^{(t)} = w_v^{(t)}$ holds for all vertices u, v with the same graph distance to s and all timesteps $t \geq 0$.*

Proof. Due to the choice of $w^{(0)}$, the claim is trivially fulfilled for $t = 0$. Following from the level structure of G, the FOS/C iteration formula for vertex v and timestep $t+1$ can be rewritten as

$$w_v^{(t+1)} = w_v^{(t)} + d_v - \alpha \sum_{i=0}^{\Lambda} \sum_{\{u,v\} \in E \wedge u \in L_i} w_v^{(t)} - w_u^{(t)}\,,$$

where Λ denotes the number of levels w. r. t. s. Now the claim follows by induction because the flow between levels $i-1$ and i and also i and $i+1$ is for all edges between the respective two levels equal in timestep t. Since also the number of edges connecting the same two levels is the same for each vertex (Lemma 3.28), each term contributes the same amount of load to $w_v^{(t+1)}$. □

We know by Proposition 3.9 that for each vertex $v \in V \backslash \{s\}$ of an arbitrary graph there exists a path from v to s such that by traversing it, the load amount increases. Now we can show that for distance-transitive graphs this property holds on *every shortest* path.

Theorem 3.31. *If G is distance-transitive, then for all $u, v \in V$ with $\mathrm{dist}(u,s) < \mathrm{dist}(v,s)$ it holds that $[w]_u^s > [w]_v^s$.*

Proof. Recall the equivalence of the FOS/C convergence state to the $\|\cdot\|_2$-minimal flow problem of Observation 3.6. When load is sent from node s to a node $v \in L_i$, this load has to pass all levels $L_{i'}$ with $i' < i$, $0 < i \leq \Lambda$. Shortcuts are not possible due to the properties of the level structure. This means that at least one vertex $v' \in L_{i-1}$ exists with a positive flow of load towards $v \in L_i$. Since $f_{\{v',v\}} = [w]_{v'}^s - [w]_v^s$, v' must have a higher load than v. Due to the load equality in level $i-1$ (Proposition 3.30), all vertices of level $i-1$ have a higher load than vertices in level i. □

Note that, although the order induced by the FOS/C diffusion distance corresponds to the one induced by the ordinary graph distance, the load differences across levels reflect their connectivity, so that FOS/C still reveals more information. This becomes clear in the following, where we derive alternative representations of the convergence flow f. Once the flow is known, the loads can be deduced from it.

Lemma 3.32. *Let $G = (V, E)$ be a distance-transitive graph, $s \in V$, and $e = \{u, v\} \in E$ with $u \in L_i(s)$ and $v \in L_{i+1}(s)$ $(0 < i < \Lambda)$. Then, the number of shortest paths starting in s and ending in level i is equal for all vertices in L_i.*

Similarly, let $V' := \bigcup_{j=i+2}^{\Lambda} L_j$ be the subset of nodes further away from s than v. Then, there are exactly as many shortest paths from s to V' via v as via any other vertex in level $i+1$.

Proof. We prove the first claim by induction on the level number i: For $i = 1$, there is exactly one respective shortest path from s to any arbitrary vertex $u' \in L_1$. Assuming now the claim to be true for all $i' \leq i$, let $v' \neq v$ be a vertex in level $i+1$. The number of shortest paths to the neighbors in level i of v' is the same as for the neighbors in level i of v. Moreover, the number of neighbors is also the same for v and v' (Lemma 3.28). Since the edges $\{u, v\}$ and $\{u', v'\}$ (with $u' \in L_i$) are disjoint, the claim follows also for level $i+1$.

For a similar proof of the second claim, let $\overline{v} \in L_{i+1}$ with $\overline{v} \neq v$. As $\overline{v}$ and v are reached by the same number of shortest paths and they have the same number of neighbors in L_{i+2}, level L_{i+2} may serve as induction basis. We know by Lemma 3.28 that the nodes in

level L_j have the same number of neighbors in level L_{j+1}, $i+2 \leq j < \Lambda$. The respective edges running between nodes of consecutive levels are each part of a new shortest path. Hence, an inductive step from j to $j+1$ adds the same number of shortest paths from s to L_{j+1} via v and via $\overline{v}$. Combined with the first part, we obtain that v and $\overline{v}$ are crossed by the same number of shortest paths from s to V'. □

Corollary 3.33. *Let G and e be defined as in Lemma 3.32. Then e lies on the same number of shortest paths from s to nodes in $L_j(s)$, $i < j \leq \Lambda$, as any other edge $e' = \{u', v'\}$ with $u' \in L_i(s)$ and $v' \in L_{i+1}(s)$.*

Theorem 3.34. *Let G and s be defined as in Lemma 3.32 and let $E_{i,i+1}(s) := \{\{u,v\} : u \in L_i,\, v \in L_{i+1}\}$ denote the set of edges running between levels i and $i+1$, $0 \leq i < \Lambda$. Then, the FOS/C convergence flow f_e on an edge $e = \{u,v\} \in E_{i,i+1}(s)$ is given by*

$$w_u - w_v = f_e = \frac{\delta}{|E_{i,i+1}(s)|} \cdot \sum_{j=i+1}^{\Lambda} |L_j| \,.$$

Proof. We have seen before that the load which reaches $u \in L_i$ needs to pass all levels $L_{i'}$ with $i' < i$. Moreover, nodes of the same level have the same load (Proposition 3.30). Consequently, edges running between the same two levels get the same amount of flow since the flow is the load difference. As all nodes in levels larger than i need to receive their load amount δ, the total amount of load crossing the edges of $E_{i,i+1}(s)$ is $\delta \sum_{j=i+1}^{\Lambda} |L_j|$, which has to be divided by the number of edges to obtain the convergence flow on a single edge $e \in E_{i,i+1}(s)$. □

Since we can determine the size of each level and the number of edges between different levels by simple breadth-first-search (BFS) techniques, the FOS/C convergence flow f can be computed on distance-transitive graphs without solving a linear system. While BFS is asymptotically not faster than an optimal linear solver, the constants involved in the running time of BFS can be expected to be much smaller.

Each node of the κ-dimensional hypercube $Q(\kappa)$ corresponds to a bit-string of length κ. Since Q(κ) is κ-regular, vertex- and edge-transitive [Bigg 93], we may assume w. l. o. g. that $s = 0^\kappa$. Due to its known structure, the FOS/C convergence flow on the hypercube can be stated more explicitly and the monotonicity result can be further improved.

Corollary 3.35. *On the κ-dimensional hypercube $Q(\kappa) = (V,E)$, the FOS/C convergence flow f_e on an edge $e = \{u,v\} \in E$ (u in level i, v in level $i+1$, $0 \leq i < \Lambda$) is $w_u - w_v = f_e = \frac{\delta}{\binom{\kappa}{i}(\kappa - i)} \cdot \sum_{j=i+1}^{\kappa} \binom{\kappa}{j}$.*

Proof. Since one chooses i out of κ bits to be set to 1 to reach a level-i vertex, level i of $Q(\kappa)$ contains $\binom{\kappa}{i}$ vertices. Consequently, $|E_{i,i+1}(s)| = \binom{\kappa}{i}(\kappa - i)$, as each node in level i has $\kappa - i$ neighbors in level $i+1$. □

Theorem 3.36. *For all nodes $u, v \in V$ of $Q(\kappa) = (V, E)$, the monotonicity result of Theorem 3.31 holds in all timesteps $t \geq 0$, not only in the convergence state: If s is the source node (w. l. o. g. $s = 0^\kappa$) and $\mathrm{dist}(u, s) < \mathrm{dist}(v, s)$, $[w^{(t)}]_u^s > [w^{(t)}]_v^s$ for all $t \geq 0$.*

Proof. The claim is proved by induction on t. Due to Theorem 3.30 we can denote the load of a vertex in level l, $1 \leq l < \kappa$, and timestep t by $w_l^{(t)}$. Thus, we rewrite the update procedure of FOS/C on $Q(\kappa)$ as

$$w_l^{(t)} := w_l^{(t-1)} - \delta + l\alpha(w_{l-1}^{(t-1)} - w_l^{(t-1)}) + (\kappa - l)\alpha(w_{l+1}^{(t-1)} - w_l^{(t-1)})\,.$$

Note that, by restricting l to lie between 1 and $\kappa - 1$, the indices of w lie all between 0 and κ. Whenever in the following indices of w are not in the interval $[0, \kappa]$, the corresponding load values are 0. If u is in level zero (which means it equals the source node), the claim can be shown analogous to the second part of the proof of Theorem 3.46.

The claim holds for $t = 0$ due to the structure of $w^{(0)}$ (only s has positive load, all other vertices have zero load). Assuming that the claim holds for all $t' \leq t$, we can deduce for all κ and $\alpha < (\kappa + 1)^{-1}$:

$$\begin{aligned}
& w_l^{(t)} \geq w_{l+1}^{(t)} \\
\Leftrightarrow\ & w_l^{(t)}(1 - (\kappa+1)\alpha) \geq w_{l+1}^{(t)}(1 - (\kappa+1)\alpha) \\
\Leftrightarrow\ & w_l^{(t)}(1 - \kappa\alpha) + w_{l+1}^{(t)}\alpha \geq w_{l+1}^{(t)}(1 - \kappa\alpha) + w_l^{(t)}\alpha \\
\overset{IH}{\Rightarrow}\ & w_l^{(t)}(1 - \kappa\alpha) + w_{l+1}^{(t)}\alpha + w_{l-1}^{(t)}l\alpha > w_{l+1}^{(t)}(1 - \kappa\alpha) + w_l^{(t)}\alpha + w_l^{(t)}l\alpha\,.
\end{aligned}$$

Like the previous step, the following one uses the induction hypothesis. Note that even if level $l + 2$ does not exist, this step is feasible. Adding 0 on the right side of the inequation does not change its validity. Some more additions and rearranging yield

$$\begin{aligned}
\overset{IH}{\Rightarrow}\ & w_{l-1}^{(t)}l\alpha + w_l^{(t)}(1 - \kappa\alpha) + w_{l+1}^{(t)}\alpha + w_{l+1}^{(t)}(\kappa - l - 1)\alpha > \\
& w_l^{(t)}(l+1)\alpha + w_{l+1}^{(t)}(1 - \kappa\alpha) + w_{l+2}^{(t)}(\kappa - l - 1)\alpha \\
\Leftrightarrow\ & w_{l-1}^{(t)}l\alpha + w_l^{(t)}(1 - \kappa\alpha) + w_{l+1}^{(t)}(\kappa - l)\alpha \geq \\
& w_l^{(t)}(l+1)\alpha + w_{l+1}^{(t)}(1 - \kappa\alpha) + w_{l+2}^{(t)}(\kappa - l - 1)\alpha \\
\Leftrightarrow\ & w_l^{(t)} + l\alpha(w_{l-1}^{(t)} - w_l^{(t)}) - \kappa\alpha(w_l^{(t)} - w_{l+1}^{(t)}) + l\alpha(w_l^{(t)} - w_{l+1}^{(t)}) \geq \\
& w_{l+1}^{(t)} + \alpha(w_l^{(t)} - w_{l+2}^{(t)} - w_{l+1}^{(t)} + w_{l+2}^{(t)}) - (\kappa - l - 1)\alpha(w_{l+1}^{(t)} - w_{l+2}^{(t)}) \\
& + l\alpha(w_l^{(t)} - w_{l+1}^{(t)}) \\
\Leftrightarrow\ & w_l^{(t)} + l\alpha(w_{l-1}^{(t)} - w_l^{(t)}) + (\kappa - l)\alpha(w_{l+1}^{(t)} - w_l^{(t)}) - \delta \geq \\
& w_{l+1}^{(t)} + (l+1)\alpha(w_l^{(t)} - w_{l+1}^{(t)}) + (\kappa - l - 1)\alpha(w_{l+2}^{(t)} - w_{l+1}^{(t)}) - \delta \\
\Leftrightarrow\ & w_l^{(t+1)} \geq w_{l+1}^{(t+1)}\,.
\end{aligned}$$

□

Theorem 3.37. *Let G be a distance-transitive graph. Then, the uniform flow distribution of Definition 3.29 yields the $\|\cdot\|_2$-minimal FOS/C convergence flow on G.*

Proof. Due to monotonicity (Theorem 3.31) we know that the FOS/C convergence flow on an edge always directs from s away. Recall that the edges $e = \{u, v\}$ and $e' = \{u', v'\}$ with $u, u' \in L_i$ and $v, v' \in L_{i+1}$ are part of the same number of shortest paths (Corollary 3.33). Hence, the uniform flow distribution yields an even division of the flow between two levels. Such an even division is also obtained in the FOS/C convergence state (Theorem 3.34). Moreover, the amount of flow that has to reach the vertices of all levels L_j, $j \geq i$, $1 \leq i \leq \Lambda$, in the FOS/C convergence state is known. It is just the number of vertices in all such levels times δ. Hence, the amount of flow passing a level is fixed, so that the uniform distribution of the flow and the $\|\cdot\|_2$-minimal FOS/C convergence flow coincide. □

It would be nice to find such a simple characterization of the convergence flow for more graph classes. On the other hand, simple can also mean that not much more information on the structure is provided than with the ordinary graph distance. For distance-transitive graphs it tells us at least something about how well-connected the different levels are in terms of the similarity measure FOS/C.

3.4. FOS/C on the Torus

We have seen that the FOS/C convergence flow equals the uniform flow distribution on distance-transitive graphs. In this section we show that this property does not hold for the torus in general, despite the numerous torus symmetries such as vertex-transitivity. We also analyze other properties of the load distribution, in particular during the FOS/C iteration, not only in the convergence state. Note that we consider again only single-source FOS/C procedures and unweighted graphs in this section.

Definition 3.38. The κ-dimensional *torus* $T[d_1, \ldots, d_\kappa] = (V, E)$ is defined as:

$$
\begin{aligned}
V &= \{(u_1, \ldots, u_\kappa) \mid -\lfloor \frac{d_\nu - 1}{2} \rfloor \leq u_\nu \leq \lceil \frac{d_\nu - 1}{2} \rceil \text{ for } 1 \leq \nu \leq \kappa\} \text{ and} \\
E &= \{\{(u_1, \ldots, u_\kappa), (v_1, ..., v_\kappa)\} \mid \exists\, 1 \leq \mu \leq \kappa \text{ with} \\
&\quad (v_\mu = u_\mu + 1 \vee (v_\mu = -\lfloor \frac{d_\mu - 1}{2} \rfloor \wedge u_\mu = \lceil \frac{d_\mu - 1}{2} \rceil)) \text{ and } u_\nu = v_\nu \text{ for } \nu \neq \mu\}\,.
\end{aligned}
$$

Remark 3.39. A torus graph $G = (V, E)$ is vertex-transitive. This can be verified by showing that every κ-dimensional, integral translation vector is an automorphism. Alternatively, one can use the fact that a cycle graph C_n is a Cayley graph for the cyclic group $\mathbb{Z}_n$. As a κ-dimensional torus is the graph product of κ cycle graphs, it is also a Cayley graph, which is a class of vertex-transitive graphs (comp. [Gros 04, p. 505f.]).

Torus graphs are very important in theory [Leig 92] and practice [The 02], e. g., because they have bounded degree, are regular and vertex-transitive. They also correspond

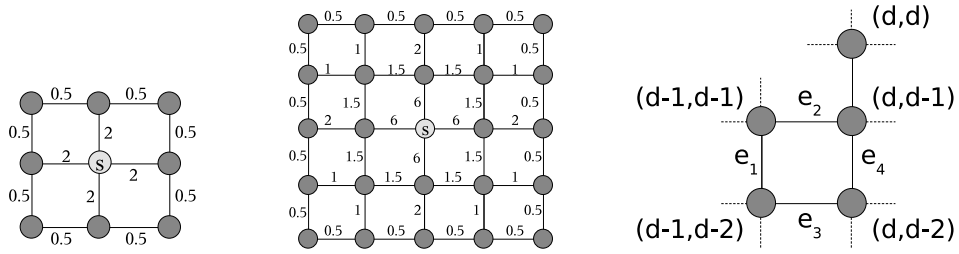


Figure 3.2.: Convergence flow of FOS/C on tori of size (a) 3x3 and (b) 5x5 with $\delta = 1$. (c) Illustration of the square made of the edges e_1, e_2, e_3, and e_4 as in the proof of Theorem 3.40.

to numerical simulation problems that decompose their domain by structured grids with cyclic boundary conditions.

In the following theorem we show that the uniform flow distribution among the shortest paths is not $\|\cdot\|_2$-optimal on the torus in general. The intuitive reason is that the number of shortest paths from a source s to another vertex u does not depend on its distance to s alone.

Theorem 3.40. *The uniform flow distribution on the 2D torus yields the $\|\cdot\|_2$-minimal flow for $d_1 = d_2 \in \{2, 3, 4, 5\}$, but not for odd $d_1 = d_2 \geq 7$.*

Proof. The FOS/C convergence flows for torus graphs of size 3×3 and 5×5 are depicted in Figures 3.2(a) and 3.2(b). (Note that the lines with only one nodal endpoint denote wraparound edges.) One can easily verify that the claim holds for these two instances and that the tori of size 2×2 and 4×4 are isomorphic to the hypercubes $Q(2)$ and $Q(4)$, respectively. Recall that the convergence flow distribution of hypercubes has been shown to be uniform.

Hence, we examine a $d_1 \times d_2$-torus with $d_1 = d_2$ odd and not smaller than 7. Intuitively, the property does not hold for larger tori because near the diagonal there are more shortest paths than on an axis. Thus, by rerouting some of the uniform flow towards the diagonal, the costs can be reduced. We proceed by setting $d := \frac{d_1 - 1}{2}$ and assuming w. l. o. g. that $s = (0, 0)$. Consider now the square consisting of the nodes $(d-1, d-2), (d, d-2), (d, d-1), (d-1, d-1)$, see Figure 3.2(c). Denote – with a slight abuse of notation – the following edges as well as the flow on them by

$$\begin{aligned} e_1 &= \{(d-1, d-2), (d-1, d-1)\}, \\ e_2 &= \{(d-1, d-1), (d, d-1)\}, \\ e_3 &= \{(d-1, d-2), (d, d-2)\}, \text{ and} \\ e_4 &= \{(d, d-2), (d, d-1)\}. \end{aligned}$$

For the $\|\cdot\|_2$-minimal flow it is necessary that $e_1 + e_2 = e_3 + e_4$. Otherwise, some flow from node $(d-1, d-2)$ to node $(d, d-1)$ could be rerouted to decrease the costs.

Observe that in the uniform flow distribution the amount of flow routed via e_4 is either sent to $(d, d-1)$ or to (d, d). The amount on e_4 destined alone to $(d, d-1)$ is the quotient of the number of shortest paths to $(d, d-1)$ via $(d, d-2)$ and the number of all shortest paths to $(d, d-1)$. It is well-known that the number of shortest paths from $(0, 0)$ to (x, y) on the torus is $\binom{x+y}{x} = \binom{x+y}{y}$. Thus, by applying the same quotient argument for the flow to (d, d), we obtain

$$e_4 = \binom{2d-2}{d-2} \cdot \left(\frac{1}{\binom{2d-1}{d-1}} + \frac{1}{\binom{2d}{d}} \right) \text{ and } e_2 = \binom{2d-2}{d-1} \cdot \left(\frac{1}{\binom{2d-1}{d-1}} + \frac{1}{\binom{2d}{d}} \right) \geq e_4$$

because $\binom{n}{k}$ is maximized for $k = n/2$. It remains to be shown that $e_1 > e_3$ holds. Similar as before, we have

$$e_1 = \binom{2d-3}{d-1} \cdot \left(\frac{1}{\binom{2d-2}{d-1}} + \frac{1}{\binom{2d-1}{d-1}} + \frac{1}{\binom{2d-1}{d-1}} + \frac{2}{\binom{2d}{d}} \right),$$

$$e_3 = \binom{2d-3}{d-1} \cdot \left(\frac{1}{\binom{2d-2}{d-2}} + \frac{1}{\binom{2d-1}{d-1}} + \frac{1}{\binom{2d}{d}} \right).$$

Some rearranging yields

$$\frac{e_1 - e_3}{\binom{2d-3}{d-1}} = \frac{(3d^2 - 7d + 2)(d!)^2}{d(d-1)(2d!)}.$$

Provided that $d > 2$, we have: $3d^2 - 7d + 2 = 3d \cdot d - 7d + 2 \geq 9d - 7d + 2 = 2d + 2 > 0$. Thus, $e_1 - e_3 > 0$, implying the claim. □

We finish the results on the uniform flow distribution by the following proposition. Note that, since we have shown above that the uniform flow is not $\|\cdot\|_2$-minimal for general torus graphs, its implication is not an equivalence.

Proposition 3.41. *If on a graph $G = (V, E)$ the uniform flow distribution is $\|\cdot\|_2$-minimal, then for $\{u, v\} \in E$ and $\operatorname{dist}(u, s) < \operatorname{dist}(v, s)$ it holds that $w_u > w_v$.*

Proof. Since load is routed uniformly via shortest paths, we know that some load is routed over every shortest path. Therefore, as u is on a shortest path from s to v, the load difference (which is the flow) between u and v must be positive. □

Now set $\alpha := (\deg(G) + 1)^{-1}$, so that all entries of the diffusion matrix $\mathbf{M}$ are either 0 or α. This is a usual choice for transition matrices in random walk theory. Consider an arbitrary κ-dimensional torus $T[d_1, \ldots, d_\kappa]$. Due to vertex-transitivity, we may assume w. l. o. g. that the source node s is the zero-vector.

Lemma 3.42. *[Alon 00, p. 151] For vertex-transitive graphs G, all automorphisms φ, and all timesteps t it holds:* $[\mathbf{M}^t]_{u,v} = [\mathbf{M}^t]_{\varphi(u),\varphi(v)}$.

Observation 3.43. *From this result by Alon and Spencer it follows that on vertex-transitive graphs the FOS/C load vector underlies the same permutation induced by an automorphism φ as the variables of $\mathbf{M}$. Hence, all single-source FOS/C procedures yield a permutation of the same load vector on these graphs.*

Using Lemma 3.42 and its statement on automorphisms, the following theorem regarding the monotonicity of the diffusion load can be derived.

Theorem 3.44. *[Meye 06b] Let $T[d_1, \ldots, d_\kappa] = (V, E)$, κ arbitrary, be a torus graph. For $\alpha = (\deg(G) + 1)^{-1}$ and all adjacent nodes $u, v \in V$ distinct from $s \in V$ with $\operatorname{dist}(u, s) = \operatorname{dist}(v, s) - 1$ it holds:*

$$\forall t \in \mathbb{N}_0 : [\mathbf{M}^t]_{u,s} \geq [\mathbf{M}^t]_{v,s}\,.$$

Note that one can show with a modified three-dimensional hypercube as a counterexample that this monotonicity does not hold for all vertex-transitive graphs in all timesteps. Furthermore, the general result $[\mathbf{M}^{2t}]_{u,u} \geq [\mathbf{M}^{2t}]_{u,v}$ for random walks without loops on vertex-transitive graphs can be found in Alon and Spencer [Alon 00, p. 150]. It is improved significantly on torus graphs by Theorem 3.44.

Lemma 3.45. *Assume w. l. o. g. that the source node is the origin $(0, \ldots, 0)$. Then, node $u = (u_1, \ldots, u_\kappa)$ has the same load as $\varphi(u)$, where φ is an automorphism that reflects any of the κ coordinates of u at the corresponding middle axis. Hence, the load distribution is symmetric w. r. t. all the axes and the origin of T.*

Proof. Due to Lemma 3.42 all which remains to be shown is that a map $\varphi_i(u_1, \ldots, u_\kappa) \mapsto (u_1, \ldots, u_{i-1}, -u_i, u_{i+1}, \ldots, u_\kappa)$, which performs the described reflection w. r. t. to the middle axis of dimension i, is an automorphism. The remainder of the claim (the symmetry w. r. t. the origin) then follows because it can be expressed as a concatenation of automorphisms. Such a concatenation is again an automorphism (cf. Definition 3.24). Note that in case d_i is even, the nodes with the highest distance in dimension i are mapped onto themselves: $\varphi_i(u_1, \ldots, u_{i-1}, \frac{d_i}{2}, u_{i+1}, \ldots, u_\kappa) \mapsto (u_1, \ldots, u_{i-1}, -u_i, u_{i+1}, \ldots, u_\kappa) = (u_1, \ldots, u_{i-1}, \frac{d_i}{2}, u_{i+1}, \ldots, u_\kappa)$.

Since φ_i is a bijection, it suffices to show that $\{u, v\} \in E \Rightarrow \{\varphi_i(u), \varphi_i(v)\} \in E$ holds. Let v be a neighbor of u in the j-th dimension: $v = (u_1, \ldots, u_{j-1}, u_j \pm 1, u_{j+1}, \ldots, u_\kappa)$. (The use of wrap-around edges for the neighbor relation can be handled by an appropriate use of the modulo function.) Then, if $i = j$, we have $\varphi_i(v) = (u_1, \ldots, u_{j-1}, -u_i \pm 1, u_{j+1}, \ldots, u_\kappa)$, which is obviously a neighbor of $\varphi_i(u)$. Otherwise, i. e., if $i \neq j$ and w. l. o. g. $i > j$, then $\varphi_i(v) = (u_1, \ldots, u_{j-1}, u_j \pm 1, u_{j+1}, \ldots, u_{i-1}, -u_i, u_{i+1}, \ldots, u_\kappa)$. Consequently, $\varphi_i(v)$ is again a neighbor of $\varphi_i(u)$, proving the claim. □

Following from Lemma 3.45, the FOS/C load distribution on a torus and a grid graph are equal in all timesteps if their d_i are all odd and s is located at the center of the graphs. In this case the torus is not different from the grid because there is no flow via its wraparound edges: The incident nodes of these edges have the same load, which results in a zero flow. Finally, the monotonicity result of Theorem 3.44 can be refined as follows.

Theorem 3.46. *Let the torus T and its vertices s, u, v be defined as in Theorem 3.44.*

1. $\forall t < \operatorname{dist}(u,s) : [w^{(t)}]^s_u = [w^{(t)}]^s_v = -t\delta$, $\forall t \in \{\operatorname{dist}(u,s), \ldots, \infty\} : [w^{(t)}]^s_u > [w^{(t)}]^s_v$.
2. *Let $d_\nu \geq 4$ for all $1 \leq \nu \leq \kappa$. Then: $[w^{(t)}]^s_s > [w^{(t)}]^s_{v'}$ for all timesteps t and all $v' \in V \backslash \{s\}$.*

Proof. The first claim follows directly by combining Corollary 3.14 and Theorem 3.44. For the second claim let u' denote one arbitrary (but fixed) neighbor of s. Other nodes than the neighbors need not be considered due to the monotonicity established in the first part. The remainder of the proof uses induction on the timestep t. For $t = 0$ the claim is trivially fulfilled due to the choice of $w^{(0)}$. Assume now that the claim is true for timestep t. Then, with $\Delta := \operatorname{maxdeg}(G)$, $\alpha = (\Delta + 1)^{-1}$, and the fact that G is regular:

$$\begin{aligned}
[w^{(t+1)}]^s_s - [w^{(t+1)}]^s_{u'} &= [w^{(t)}]^s_s + (n-1)\delta - \alpha\Big(\sum_{\{s,u\}\in E} [w^{(t)}]^s_s - [w^{(t)}]^s_u\Big) \\
&\quad - \Big([w^{(t)}]^s_{u'} - \delta - \alpha\Big(\sum_{\{u',v\}\in E} [w^{(t)}]^s_{u'} - [w^{(t)}]^s_v\Big)\Big) \\
&= n\delta + (1 - \alpha\Delta)([w^{(t)}]^s_s - [w^{(t)}]^s_{u'}) \\
&\quad + \alpha\Big(\sum_{\{s,u\}\in E} [w^{(t)}]^s_u - \sum_{\{u',v\}\in E} [w^{(t)}]^s_v\Big).
\end{aligned}$$

Extracting $[w^{(t)}]^s_s$ and $[w^{(t)}]^s_{u'}$ from their respective sums and $(1 - \alpha\Delta) = \alpha$ yield

$$\begin{aligned}
[w^{(t+1)}]^s_s - [w^{(t+1)}]^s_{u'} &= n\delta + \alpha\Big([w^{(t)}]^s_s - [w^{(t)}]^s_{u'} - [w^{(t)}]^s_s + \sum_{\{s,u\}\in E,\, u\neq u'} [w^{(t)}]^s_u \\
&\quad + [w^{(t)}]^s_{u'} + \sum_{\{u',v\}\in E,\, v\neq s} [w^{(t)}]^s_v\Big) \\
&= n\delta + \alpha\Big(\sum_{\{s,u\}\in E,\, u\neq u'} [w^{(t)}]^s_u - \sum_{\{u',v\}\in E,\, v\neq s} [w^{(t)}]^s_v\Big).
\end{aligned}$$

Observe that both sums above have $\Delta - 1$ summands. Moreover, we can rewrite the sum term as follows: $\sum_{\{s,u\}\in E,\, u\neq u'} [w^{(t)}]^s_u - \sum_{\{u',v\}\in E,\, v\neq s} [w^{(t)}]^s_v = \sum_{i=1}^{\Delta-1} ([w^{(t)}]^s_{u_i} - [w^{(t)}]^s_{v_i})$ with $\{s, u_i\} \in E$, $u_i \neq u'$, $\{u', v_i\} \in E$, and $v_i \neq s$. Let w. l. o. g. $u' = (s_1, \ldots, s_{j-1}, s_j + 1, s_{j+1}, \ldots, s_\kappa)$ and choose $u'' := (s_1, \ldots, s_{j-1}, s_j - 1, s_{j+1}, \ldots, s_\kappa)$ as its reflection across s. By Lemma 3.45 both u' and u'' always have the same load, so that $[w^{(t)}]^s_{u'}$ and $[w^{(t)}]^s_{u''}$ are interchangeable.

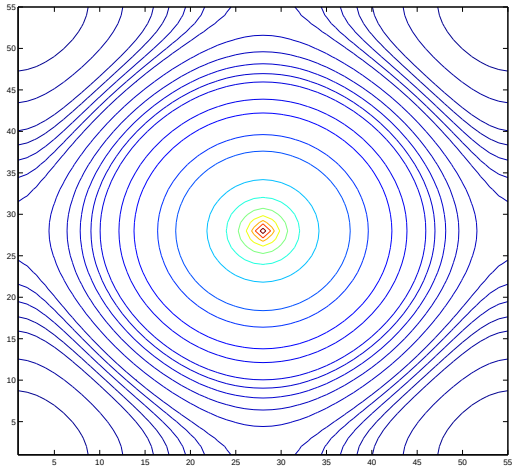


Figure 3.3.: Contour lines of FOS/C convergence load on a 2D torus.

The u_i corresponding to v_i is chosen as $u_i = (v_1, \ldots, v_{j-1}, v_j - 1, v_{j+1}, \ldots, v_\kappa)$. Hence, we subtract $[w^{(t)}]^s_{v_i}$ from the load of one of its neighbors u_i that is also a neighbor of s. By Theorem 3.44 we have $\forall t \in \mathbb{N}_0 : [\mathbf{M}^t]_{u_i,s} \geq [\mathbf{M}^t]_{v_i,s}$ and therefore $[w^{(t)}]^s_{u_i} - [w^{(t)}]^s_{v_i} \geq 0$ for each summand. Finally, this yields $[w^{(t+1)}]^s_s - [w^{(t+1)}]^s_{u'} \geq n\delta > 0$. □

A natural question in the context of tori and FOS/C is if one can characterize the FOS/C convergence load distribution more concretely. For the two-dimensional torus we have conducted experiments that reveal a certain shape of the distribution. An example is presented in Figure 3.3, which shows the contour lines of the FOS/C convergence load on $T[55, 55]$ with source $(0, 0)$. Nodes on these lines have the same amount of load.

Very close to the source the contour lines resemble a square with one corner pointing downwards. When one moves away from the source, the contour lines get a circular shape. Finally, in the corners of the torus, the contour lines become hyperbolic. Other square tori show a similar behavior. These experiments confirm results of Ellis's experiments on torus hitting times [Elli 01b]. The circular contour lines appear to be dominating in the load distribution. However, the possibility to use the wrap-around edges seems to prevent the continuation of these circular shapes in the corners of the torus.

Note that the contour lines remain circular in an asymptotic sense on an infinite grid, even if one moves far away from the source, which has been shown by Mangad:

Theorem 3.47. *[Mang 66] Let $P = (x_P, y_P)$, $Q = (x_Q, y_Q)$ be two arbitrary nodes (and their coordinates) of an infinite two-dimensional grid with mesh width h. Furthermore, let $\gamma = 0.57722\ldots$ be Euler's constant. If $\rho = \overline{PQ} = \sqrt{((x_Q - x_P)^2 + (y_Q - y_P)^2}$, then the bounds for the discrete Green's function (Laplacian pseudoinverse) $g_P(Q)$ are*

$$\frac{53 \cdot 6h^2}{\rho^2} \leq 2\pi g_P(Q) - \log \rho - \frac{3}{2}\log 2 - \gamma \leq \frac{53 \cdot 6h^2}{\rho^2} + \frac{h^2}{12\rho^2},\ \rho \geq h > 0\,.$$

Consequently, all nodes Q with the same Euclidean distance to P have the same value $g_P(Q)$ in an asymptotic sense, which leads to circular isolines.

3.5. FOS/V: FOS/C with a Virtual Vertex

After these results for specific graph classes, we turn our attention to solving FOS/C procedures on general graphs again. It has already been shown in this chapter that their convergence state can be computed by solving a linear system. Recall that this can be accomplished by fast linear solvers such as CG or algebraic multigrid. However, since $\mathbf{L}$ is semidefinite and therefore singular, significant numerical issues may arise during the execution of these algorithms. They include a relatively slow convergence or even divergence. This is due to the fact that the vectors involved need to be orthogonal to $(1,\ldots,1)^T$. In finite precision arithmetic it can happen that the vectors deviate from this orthogonality constraint. That is why we alter FOS/C slightly by introducing a virtual vertex. We obtain a new similarity measure, called FOS/V (V for virtual vertex), that circumvents the aforementioned numerical problems.

Definition 3.48. Given a graph $G = (V, E, \omega)$ and a constant $\phi > 0$, we construct a new graph $G_{ext} = (V_{ext}, E_{ext}, \omega_{ext})$ by inserting a virtual vertex $\tilde{v}$ that is connected to all other vertices by an edge of weight ϕ: $V_{ext} := V \cup \{\tilde{v}\}$, $E' := E \cup \{\{v, \tilde{v}\} \,|\, v \in V\}$ with $\omega_{ext}(e) = \omega(e)$ for all $e \in E$ and $\omega_{ext}(\{v, \tilde{v}\}) = \phi$.

The Laplacian matrix of G_{ext} has one additional row and one additional column compared to that of G. Both this row and this column contain the entry $-\phi$ everywhere except for the common diagonal entry, which is $|V| \cdot \phi$. Introducing a virtual vertex into the graph, results in a modified flow problem solved by FOS/C. This is reflected in the drain vector. In the original scheme all nodes $v \in V \backslash S$ consume a load amount of δ each. Here, it is the virtual vertex v' which consumes all load sent out by the source nodes:

$$[d_{ext}]_v \quad = \quad \begin{cases} \frac{\delta n}{|S|} & v \in S\,, \\ 0 & v \in V \backslash S\,, \\ -\delta n & v = v'\,. \end{cases}$$

The resulting linear system $\mathbf{L}_{ext} w_{ext} = d_{ext}$ would not be easier to solve because $\mathbf{L}_{ext}$ is still symmetric positive semidefinite. However, as pointed out by Kaasschieter [Kaas 88], the semidefinite property can be changed by fixing r solution values in w_{ext} to some specified value and removing the corresponding row and column from the linear system. The number r is the dimension of the null space $\{x \in \mathbb{R}^n : \mathbf{L}_{ext} x = \mathbf{0}\}$, which is 1 for $\mathbf{L}_{ext}$ (Fact 2.6). The removal of the row and column representing $\tilde{v}$ results in a symmetric positive-definite matrix whose condition can be controlled by the parameter ϕ. Note that this simple preconditioning has a meaningful interpretation by the notion of sending load to the virtual vertex. Node weights can be incorporated by setting the

weight of a virtual edge to the node weight times ϕ. In the remainder of this section, we show that the modification by inserting and deleting v' results in a convergence state that can be used in a comparable manner as that of FOS/C.

Definition 3.49. Let the matrix $\mathbf{L}'$ and the vectors w' and d' comply with their counterparts $\mathbf{L}_{ext}$, w_{ext}, and d_{ext}, respectively, except that all entries corresponding to $\tilde{v}$ have been removed.

Definition 3.50. (FOS/V) Given a graph $G = (V, E, \omega)$, a set of source nodes S, an initial load vector $w'^{(0)}$, the modified drain vector d' and a constant $\alpha' := (\text{maxdeg}(G) + \phi + 1)^{-1}$. Then, the load vector updates of the iterative FOS/V scheme can be written in matrix-vector notation as $w'^{(t)} = \mathbf{M}'w'^{(t-1)} + d'$, where $\mathbf{M}' = \mathbf{I} - \alpha'\mathbf{L}'^{-1}$.

Lemma 3.51. *The eigenvalues of* $\mathbf{M}'$ *lie in the interval* $(-1, 1)$.

Proof. The matrices $\mathbf{L}$ and $\mathbf{L}' = \mathbf{L} + \phi\mathbf{I}$ have the same eigenvectors; their eigenvalues are only shifted by ϕ: $\mathbf{L}z = \lambda z \Leftrightarrow \mathbf{L}z + \phi z = (\lambda + \phi)z \Leftrightarrow \mathbf{L}'z = (\lambda + \phi)z$. That is why the eigenvalues μ'_i of $\mathbf{M}'$ can be written as $\mu'_i = 1 - \alpha'(\lambda_i + \phi) = 1 - (\lambda_i + \phi)/(\text{maxdeg}(G) + \phi + 1)$, $1 \leq i \leq n$. Due to Remark 2.6 we know that $0 = \lambda_1 < \lambda_2 \leq \cdots \leq \lambda_n \leq 2\,\text{maxdeg}(G)$. Hence:

$$-1 < 1 - \frac{2\,\text{maxdeg}(G) + \phi}{\text{maxdeg}(G) + \phi + 1} \leq \mu'_i \leq 1 - \frac{\phi}{\text{maxdeg}(G) + \phi + 1} < 1\,.$$

□

Lemma 3.52. *FOS/V converges for any initial load vector* $w'^{(0)}$. *More precisely,* $w'^{(\infty)} = (\mathbf{I} - \mathbf{M}')^{-1}d' = \frac{(\mathbf{L}+\phi\mathbf{I})^{-1}d'}{\alpha}$ *and* $[w']^u_v = \delta n[(\mathbf{L} + \phi\mathbf{I})^{-1}]_{v,u}$.

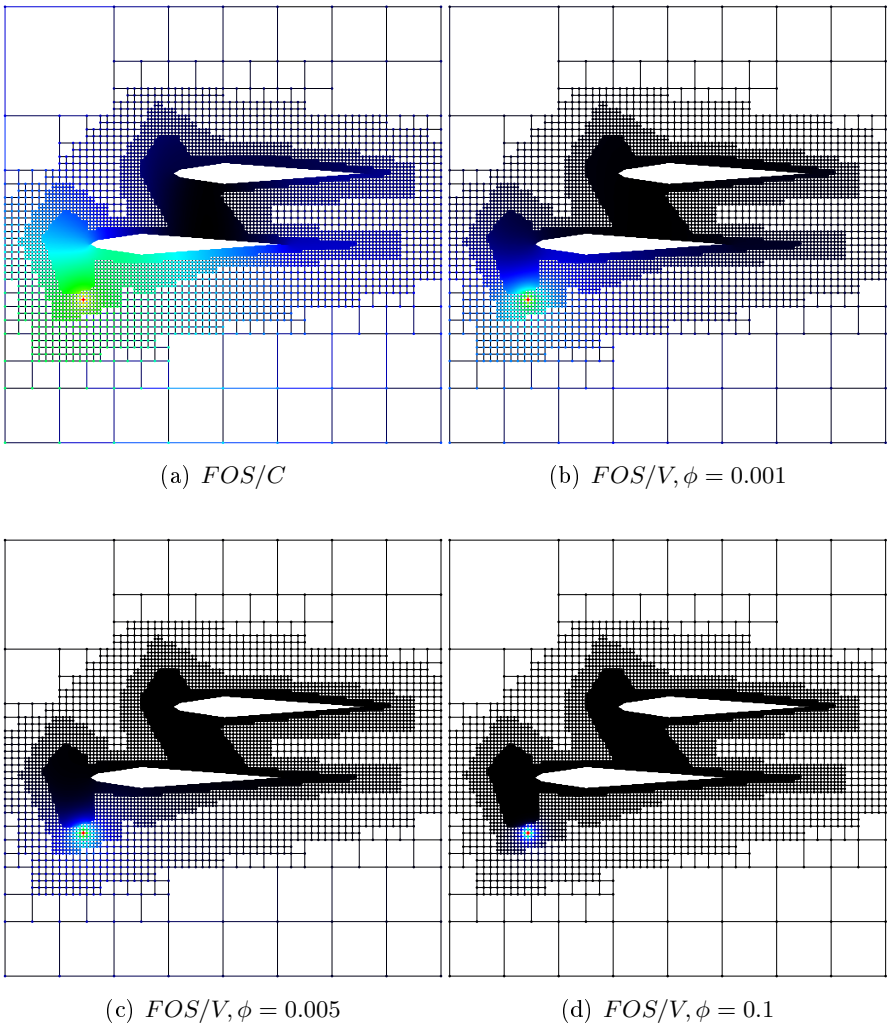
Proof. The introduction of the virtual vertex changes the expansion of the scheme only slightly compared to FOS/C:

$$w'^{(t)} \quad = \quad (\mathbf{M}')^t w'^{(0)} + ((\mathbf{M}')^{t-1} + \cdots + \mathbf{M}' + \mathbf{I})d\,.$$

Since the absolute value of the largest eigenvalue of $\mathbf{M}'$ is smaller than 1, we have $\lim_{t\to\infty}(\mathbf{M}')^t = 0$. Due to the same reason, the series $\lim_{t\to\infty}\sum_{i=0}^{t}(\mathbf{M}')^i$ converges towards $(\mathbf{I} - \mathbf{M}')^{-1} = (\alpha'\mathbf{L}')^{-1} = \alpha'^{-1}\mathbf{L}'^{-1}$ [Saad 03, p. 19], which is nonsingular.

Note that the initial load vector does not contribute to the final solution. (This behavior is not fundamentally different from before since in the scheme without virtual vertex it determines only the total load amount in the system.) Hence, $w^{(\infty)} = \sum_{i=0}^{\infty}\mathbf{M}'d = \alpha^{-1}(\mathbf{L}'^{-1}d')$. Finally, since the non-source entries of the drain vector are zero:

$$\begin{aligned}[w']^u_v &= [\mathbf{L}'^{-1}d']^u_v = \delta n[\mathbf{L}'^{-1}]_{v,u} + \sum_{u'\neq u}[\mathbf{L}'^{-1}]_{v,u'}d_{u'} \\ &= \delta n[(\mathbf{L} + \phi\mathbf{I})^{-1}]_{v,u}\,.\end{aligned}$$



(a) FOS/C (b) $FOS/V, \phi = 0.001$

(c) $FOS/V, \phi = 0.005$ (d) $FOS/V, \phi = 0.1$

Figure 3.4.: Different load distributions on the graph *biplane9* with the same source node with different ϕ. The tiny red and yellow regions indicate high load values. Green, cyan, and light blue represent medium load values, while dark blue and black indicate low load values.

□

A result of Stieltjes [Stie 86] (compare [McDo 95]) is now helpful for showing that FOS/V is also a similarity measure.

Lemma 3.53. *[Stie 86] Let* $\mathbf{A} \in \mathbb{R}^{n \times n}$ *be a symmetric, nonsingular, and diagonally dominant M-matrix, i. e., the eigenvalues of* $\mathbf{A}$ *are positive,* $\mathbf{A} \cdot \mathbf{1}$ *is a nonnegative vector, and for all* $i, j \in \{1, \ldots, n\}$*:* $a_{i,i} > 0$ *and* $a_{i,j} \leq 0$ *for* $i \neq j$*. Let* $\mathbf{C} = \mathbf{A}^{-1}$ *and fix* $i \in \{1, \ldots, n\}$*. Then* $c_{i,i} \geq c_{j,i} \geq 0$ *for all* $j \in \{1, \ldots, n\}$*.*

Corollary 3.54. *Recall from Lemma 3.52 that* $[w']_v^u = \delta n[(\mathbf{L}+\phi\mathbf{I})^{-1}]_{v,u}$*. It is well-known for a quadratic nonsingular matrix* $\mathbf{A}$ *that* $(\mathbf{A}^{-1})^T = (\mathbf{A}^T)^{-1}$ *[Bron 97, p. 244]. Moreover, it follows directly from its definition that* $\mathbf{L}+\phi\mathbf{I}$ *is a diagonally dominant M-matrix*

Table 3.1.: CG iterations needed to solve a single-source FOS/C or FOS/V procedure on different graphs.

Graph	$\|V\|$	$\|E\|$	FOS/C	$\phi = 0.001$	$\phi = 0.05$	$\phi = 0.1$
biplane9	21,701	42,038	860	579	324	78
shock9	36,476	71,290	1043	676	347	82
ocean	143,437	409,593	753	635	400	103
naca	124,799	4,162,508	242	242	239	192
dime20	224,843	336,024	4586	745	341	78

and also nonsingular. Hence, we have $[w']_v^u = [w']_u^v$ *and* $[w']_v^v \geq [w']_u^v$ *(Lemma 3.53), so that FOS/V is also a similarity measure.*

The results above show that there are two major differences between the convergence states of FOS/C and FOS/V. First, the limit of the series in the latter contains $\mathbf{M}'$ instead of $\mathbf{M}$. The second difference is the modified definition of the drain vector. These changes have one major consequence for the load vector in FOS/V: the higher ϕ is, the closer stays most of the load around the source set S. This property is visualized in Figure 3.4. There, the same disturbed diffusion problem is solved four times, each with a different ϕ, resulting in very different distributions w. r. t. the steepness of the load function. Since the load function should not be too steep for meaningful results, ϕ may not be set to arbitrarily high values. An example setting for this value and its influence on partitioning speed and quality is presented in Section 4.7. A theoretical quantification of the difference between FOS/C and FOS/V in terms of eigenvalues and -vectors is given next. It reveals why ϕ should not be too small, either.

Theorem 3.55. *The difference of FOS/C and FOS/V can be quantified as*

$$[w]_v^s - [w']_v^s = \delta n\Big(-\frac{1}{\phi} + \sum_{j=2}^{n} (\frac{1}{\lambda_j} - \frac{1}{\lambda_j + \phi})[z_j]_v [z_j]_s \Big) .$$

Proof. Recall that the matrices $\mathbf{L}$ and $\mathbf{L}'$ have the same eigenvectors. Hence, $z_1 = (1, \ldots, 1)^T$ and it follows from Corollary 3.20 and Lemma 3.52:

$$\begin{aligned}
[w]_v^s - [w']_v^s &= \delta n\Big(l_{v,s}^{\dagger} - [(\mathbf{L} + \phi \mathbf{I})^{-1}]_{v,s} \Big) \\
&= \delta n\Big(\sum_{j=2}^{n} \lambda_j^{-1} [z_j]_v [z_j]_s - \sum_{j=1}^{n} (\lambda_j + \phi)^{-1} [z_j]_v [z_j]_s \Big) \\
&= \delta n\Big(-\frac{1}{\phi} + \sum_{j=2}^{n} (\frac{1}{\lambda_j} - \frac{1}{\lambda_j + \phi})[z_j]_v [z_j]_s \Big) .
\end{aligned}$$

□

The parameter ϕ appears twice in the final term $\delta n(-\frac{1}{\phi} + \sum_{j=2}^{n} (\frac{1}{\lambda_j} - \frac{1}{\lambda_j + \phi}) z_{v,j} z_{s,j})$. Its first occurrence indicates that ϕ should not tend to zero. Otherwise, the difference to

FOS/C would go to infinity. A large ϕ, on the other hand, would yield a large difference in $(\frac{1}{\lambda_j} - \frac{1}{\lambda_j+\phi})$. Unfortunately, a good choice of ϕ needs to be determined experimentally, which is certainly a drawback of the method. Table 3.1, however, shows the effectiveness of the virtual vertex approach. For several graphs it displays the number of CG iterations necessary to solve a single-source procedure of FOS/C ($\phi = 0$) or FOS/V to a certain accuracy. Clearly, the higher ϕ becomes, the fewer iterations are necessary. Only the graph *naca* provides little room for improvement, unless ϕ is very high.

Using a virtual vertex has even more advantages than improving the convergence and robustness of iterative solvers. If several different load distributions computed by FOS/V have to be compared, the virtual vertex acts as a common reference if its load value is always fixed to 0. Thus, a normalization by fixing $\overline{w}$ is not necessary any more. Moreover, unlike in our previous work on the subject with Schamberger [Meye 05], the virtual vertex is eliminated from the actual solution process, which makes the use of multigrid/multilevel methods easier and further speeds up the computations.

4. A Shape-optimizing Partitioning Algorithm

In the previous chapter it has been shown that FOS/C is able to determine if two nodes (or regions) of a graph are densely connected with each other. This property makes it a good choice as a similarity measure within a graph clustering/partitioning algorithm. By integrating FOS/C into the Bubble framework, we facilitate that Bubble can distinguish dense from sparse regions of a graph. Moreover, the employment of FOS/C yields an implicit optimization of the subdomain shapes. This can be seen from meshes that stem from numerical simulations and have geometric information; their subdomains tend to be convex in a geometric sense.

How the integration of FOS/C into Bubble is done, is shown in this chapter. Note that FOS/V is only mentioned in the experimental section within this chapter. On a practical level there is only a small difference to FOS/C. Within an implementation one can switch easily between the two methods. Moreover, FOS/V would require the specification of another parameter, the virtual edge weight ϕ. Hence, we simply use FOS/C as a representative for both in our algorithmic considerations (and even in the name of the algorithm Bubble-FOS/C).

In its generic form, Bubble-FOS/C is suitable for graph clustering as an extension of Lloyd's k-means algorithm to graphs. Additional balancing methods make it also suitable for graph partitioning. Yet, since our clustering approach is also based on partitioning the input (in contrast to building a hierarchy of clusters by merging or division), we simply refer to the process of *partitioning* for computing a new partition Π of V, regardless of its use for graph clustering or graph (re)partitioning.

After the definition of Bubble-FOS/C and the description of extensions to the generic framework, we discuss its computational complexity. The main theoretical result of this chapter consists of a proof that Bubble-FOS/C converges. In particular, we show by means of a potential function that it stops at a local optimum. How this convergence state may look like in a special case, is examined on the two-dimensional torus in the second part of this chapter. This second part deals with practical aspects of Bubble-FOS/C, in particular its implementation and experimental outcomes. It includes the explanation of the algebraic multigrid solver assembled specifically to solve FOS/C procedures faster. In order to make Bubble-FOS/C suitable for partitioning graphs into equally sized subdomains, additional methods are described and integrated into the algorithm. That our algorithm yields good graph partitioning results, is demonstrated by extensive

Algorithm 1 GenericBubble-FOS/C $(G, k, \Pi) \rightarrow \Pi$

```
01  if Π is defined then
      /* ComputeCenters */
02    parallel for c = 1, ..., k do
03      Initialize d_c (S = π_c)
04      Solve L w_c = d_c
05      z_c = argmax_{v ∈ π_c} [w_c]_v
06  else
      /* Arbitrary initial centers */
07    Z = InitialCenters(G, k)
08  for τ = 1, 2, ... until convergence
      /* AssignSubdomain */
09    parallel for c = 1, ..., k do
10      Initialize d_c (S = {z_c})
11      Solve and normalize L w_c = d_c
12    parallel for each v ∈ V do
13      Π(v) = argmax_{c ∈ {1,...,k}} [w_c]_v
      /* ComputeCenters */
14    parallel for c = 1, ..., k do
15      Initialize d_c (S = π_c)
16      Solve L w_c = d_c
17      z_c = argmax_{v ∈ π_c} [w_c]_v
18  return Smooth(Π)
```

experiments. Following their presentation, possible improvements of Bubble-FOS/C's running time are discussed.

4.1. Generic Bubble-FOS/C Algorithm

The major two alternating operations of the Bubble framework, as described in Section 1.3.4, are the assignment of nodes to the nearest subdomain center and the computation of center nodes for each of these subdomains. Both operations require a notion of distance or similarity, which we provide in Bubble-FOS/C by FOS/C procedures, as shown in the outline of Algorithm 1. It is explained in more detail below.

Definition 4.1. Let $\Pi = \{\pi_1, \ldots, \pi_k\}$ denote the set of the current subdomains, $Z = \{z_1, \ldots, z_k\}$ the set of the current center nodes, and $\Pi^{(\tau)}$ and $Z^{(\tau)}$ their instances in iteration τ of Bubble-FOS/C (where $\tau = 0$ is assumed in the step before the loop).

4.1.1. Initial Centers

As the first step of the algorithm, pairwise disjoint initial centers (lines 1 to 7) need to be determined. If an initial partition is provided in Π, this can be done by simply performing the `ComputeCenters` operation (lines 2 to 5), which is explained in more

Algorithm 2 LoadBasedInitialCenters $(G, k) \rightarrow Z$

```
   /* Vector variables are written in bold font */
   /* Init variables and first center */
1  z = random v ∈ V; Z[1] = z; x = 0
   /* Compute remaining centers */
2  for c = 2 to k
3    Compute load vector [w]^z
4    x = x + [w]^z
5    z' = argmin_v{[x]_v}
6    Z[c] = z'
7    z = z'
8  return Z
```

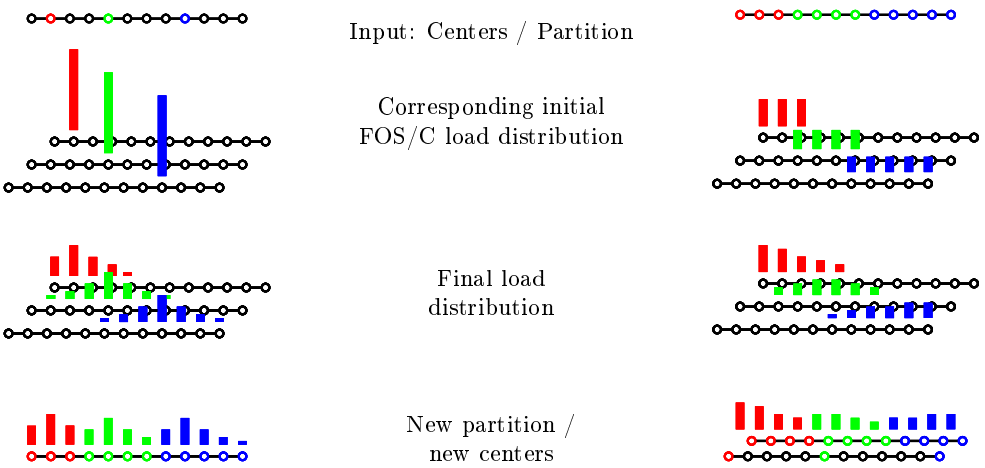


Figure 4.1.: Schematic view of `AssignSubdomain` (left) and `ComputeCenters` (right).

detail below. Otherwise, the centers are computed in an arbitrary manner (line 7). If the number of nodes in the graph equals k, each node becomes a center. Yet, if $|V| > k$, a good strategy how to compute the center nodes is important. Schamberger [Scha 06] has proposed to choose all center nodes randomly or to choose the subdomain number for each node randomly at the beginning. While these approaches are feasible, they can result in very bad initial partitions.

To avoid the choice of a very bad center set, we would like to gain more control over the distribution of the centers in the graph. Therefore, we extend previous work to the diffusion-based setting with FOS/C. As in Diekmann et al. [Diek 00], we choose only one initial node randomly, which becomes the first center. Then, to find the initial center of the next subdomain, Diekmann et al. performs a breadth-first search (BFS) from all already chosen center nodes. The node which is found last becomes the next center. The idea of choosing the next center farthest away is modified here to incorporate disturbed diffusion. We replace BFS by FOS/C for computing the next center in the procedure LoadBasedInitialCenters, see Algorithm 2. This procedure chooses the next center least similar (i. e., with minimum FOS/C loads: $\text{argmin}_v\{\sum_{z \in Z}[w]_v^z\}$) to all already

chosen centers. On a small graph, e. g., the coarsest graph of a multilevel hierarchy, the computation of the FOS/C loads is not expensive. It can even be repeated with different initial centers to have a choice from different sets of centers. By this repetition, outliers with a really poor solution quality can be avoided in most cases.

4.1.2. The Main Loop

After initial centers have been found, the two operations `AssignSubdomain` (lines 9 to 13) and `ComputeCenters` (lines 14 to 17) are alternated, until convergence is reached (line 8). Convergence can be detected, for example, by testing if the center set has changed from one iteration to the next one.

How FOS/C procedures are used and what their effect is within BUBBLE-FOS/C, is illustrated graphically for a path graph and $k = 3$ in Figure 4.1. For the operation `AssignSubdomain` (left), we are given a center node for each subdomain (top). Independently for each subdomain we compute a single-source FOS/C procedure whose source set consists only of the respective center node. After the load is spread, the k load functions defined on the nodes have a hilly shape (middle). Finally, we assign each node v to the subdomain it has obtained the highest load amount from (bottom).

For `ComputeCenters` (right) the source set is not a single node. Instead, all nodes of subdomain c belong to the source set S_c of the c-th multiple-source FOS/C procedure, $1 \leq c \leq k$. After performing the respective multiple-source FOS/C procedure independently for each subdomain (middle), the node with the highest load of each respective subdomain becomes its new center node (bottom). The rare case of ties within these two operations is handled in the following manner. If a node has received the same highest load from more than one FOS/C procedure within `AssignSubdomain`, it chooses the subdomain it already belongs to, or – if the current subdomain is not among the candidates – the one with the smallest index. In case more than one node is a candidate for the new center within `ComputeCenters`, we proceed analogously.

Boundary Smoothing

The final (optional) operation before returning the partition Π is SMOOTH (line 18). It is based on work by Schamberger [Scha 06, p. 90] and aims at a greedy reduction of external edges. This is done by moving boundary nodes to different subdomains if this reduces the number of external edges. Note that, unlike the KL heuristic, SMOOTH considers only nodes that are at a subdomain boundary when the routine is started. The order in which the moves happen, is based on priorities. For each boundary node v and each adjacent subdomain c one priority value $p_v^c := [w_{\Pi(v)}]_v - [w_c]_v$ is computed. A value p_v^c specifies how strongly node v would like to migrate to subdomain c, where a smaller value expresses a higher desire to move.

4.2. Computational Complexity and Inherent Parallelism of Bubble-FOS/C

The complexity of the Bubble-FOS/C algorithm is dominated by the running time of the for-loop over τ. Each FOS/C based operation within this loop requires the solution of k linear systems. Since we assume these systems to be sparse (i. e., $m = \mathcal{O}(n)$), we can apply sparse linear solvers such as conjugate gradient (CG). (For an overview on sparse linear solvers see, e. g., Saad's textbook [Saad 03].) As an iterative solver CG has subquadratic runtime, where the actual convergence rate depends on the spectral condition number of the matrix. For matrices stemming from numerical simulations, it is known that CG typically requires $\mathcal{O}(n^{3/2})$ operations for 2D problems and $\mathcal{O}(n^{4/3})$ for 3D problems [Shew 94]. Algebraic multigrid can even achieve linear running time for certain matrices that model two-dimensional numerical problems [Ster 06]. Yet, for classes of matrices not covered by classical AMG theory, extensive fine-tuning is often needed to obtain linear or close to linear running time.

In any case, every iteration of the loop requires $\mathcal{O}(k\hat{c})$ operations, where $\hat{c}$ denotes the cost for solving one linear system, which depends on the solver, as noted above. The total running time of Bubble-FOS/C is then $\mathcal{O}(\hat{\tau}k\hat{c})$, where $\hat{\tau}$ denotes the total number of iterations. By employing a multilevel approach (cf. Section 4.6), convergence can be observed in practice after $\hat{\tau} = \mathcal{O}(1)$ iterations. In the optimal case, where $\hat{\tau}$ is constant and $\hat{c}$ linear, this results in $\mathcal{O}(kn)$. While the linear dependence on n is optimal, the factor k is not desirable.

For both operations `AssignPartition` and `ComputeCenters` it is necessary to solve k FOS/C procedures, one for each subdomain. Put it a different way, for each node one computes k load values. Observe that the k procedures belonging to the same operation are independent from each other. Hence, the order in which the k load values of a node v are computed, is totally irrelevant. Their solutions can be computed concurrently, for example by k parallel processors. Furthermore, once the load values are computed for all nodes, their evaluation also contains parallelism. While the assignment of nodes to subdomains is independent for each node, the center determination is independent for each subdomain. The fact that most computations of the Bubble-FOS/C algorithm can be made concurrently, makes it predestined for parallel execution. Unlike the KL/FM partitioners, it does not require complex routines to ensure data consistency. Instead, simple synchronizations that guarantee the completion of all required computations are sufficient.

Regarding memory consumption, Bubble-FOS/C has a drawback that should not be unmentioned. Since we calculate k load values per node, the total amount of memory required is $\mathcal{O}(k \cdot n)$, a factor of k higher than the input size. This high consumption is certainly undesirable, no matter if Bubble-FOS/C is executed sequentially or in parallel. A method to reduce the required memory size as well as the running time, is discussed in Section 4.8.

4.3. Convergence and Connectedness Results on Bubble-FOS/C

4.3.1. Convergence towards a Local Optimum

It is well-known that Lloyd's geometric k-means algorithm converges to a local optimum [Seli 84]. That a very similar convergence property also holds for our algorithm Bubble-FOS/C depicted as Algorithm 1, is proved in this section. The proof relies on the FOS/C load symmetry property and a potential function, for which the algorithm finds a local maximum. This proof and its potential function provide a solid characterization of our algorithm and the solutions it computes.

Definition 4.2. Let the function $F(\Pi, Z, \tau)$ be defined as follows:

$$F(\Pi, Z, \tau) := \sum_{c=1}^{k} \sum_{v \in \pi_c^{(\tau)}} [w]_v^{z_c^{(\tau)}} .$$

Note that the basic structure of F resembles the objective function in k-means cluster analysis [Seli 84]. Yet, here we adapt it to the graph setting by replacing the squared Euclidean distance to the nearest center by the highest FOS/C load for each node, which also necessitates its maximization instead of its minimization.

Since it is obvious that F has a finite upper bound on any finite graph G, it remains to be shown that F is increased by every Bubble-FOS/C operation, until convergence is reached. For this we show in the following that the two operations `AssignPartition` and `ComputeCenters` each maximize the value of F w. r. t. their input.

Lemma 4.3. *The partition computed by `AssignPartition` maximizes the value of F for a given set of centers Z.*

Proof. Applying `AssignPartition` means to fix the set Z and compute k single-source FOS/C procedures. Since each node contributes exactly one of its k load values to F and chooses for this the maximum one, F is maximized for fixed Z. □

To show the analogous property for `ComputeCenters`, it is crucial to use the load symmetry between the sources of two single-source FOS/C procedures (Proposition 3.15).

Lemma 4.4. *The set of center nodes determined by `ComputeCenters` maximizes the value of F for a given partition Π.*

Proof. Let $\Pi = \pi_1 \dot\cup \ldots \dot\cup \pi_k$ be the current partition. `ComputeCenters` solves for each subdomain π_c, $c \in \{1, \ldots, k\}$, a multiple-source FOS/C procedure, where the whole respective subdomain acts as source. Consider one of these subdomains π_c and its multiple-source FOS/C procedure with respective drain vector d and $S := \pi_c$. The FOS/C procedure solves $\mathbf{L}w = d$ for w. Our aim is now to split this procedure into subprocedures

that solve $\mathbf{L}w_i = d_i$ for w_i, $i \in S$, and that satisfy $\sum_{i \in S} d_i = d$. Note that w_i and d_i denote vectors here, not vector entries. Such a splitting

$$\mathbf{L}w = d \Leftrightarrow \mathbf{L}(w_1 + w_2 + \cdots + w_{|S|}) = d_1 + d_2 + \cdots + d_{|S|}$$

indeed exists because $\mathbf{L}$ is a linear operator. Each subprocedure $\mathbf{L}w_i = d_i$ can be defined as an ordinary *single-source* diffusion procedure. The only difference is that it is scaled by $\frac{1}{|S|}$, i. e., the drain constant is $\frac{\delta}{|S|}$ instead of δ and the total drain is $\frac{\delta n}{|S|}$ instead of δn. Entry v of the drain vector d_i is then

$$[d_i]_v = \begin{cases} \frac{\delta n}{|S|} - \frac{\delta}{|S|} & v \in S,\, v \text{ source of subprocedure } i\,, \\ -\frac{\delta}{|S|} & \text{else}\,. \end{cases}$$

The perpendicularity property $d_i \perp (1, \ldots, 1)^T$ holds for all $i \in S$ since the sum of all entries in each vector d_i equals $\frac{\delta n}{|S|} - \frac{\delta}{|S|} + (n-1) \cdot \frac{-\delta}{|S|} = 0$. Thus, each subprocedure has a solution. This solution can again be normalized to be unique and comparable (e. g., by the additional constraint $w_i \perp (1, \ldots, 1)^T$ for all i). Moreover, $\sum_{i \in S} d_i = d$ holds as well: For $v \in V \backslash S$ we have: $\sum_{i \in S} [d_i]_v = -\delta$ and for $v \in S$ the sum evaluates to: $\sum_{i \in S} [d_i]_v = \frac{\delta n}{|S|} - \frac{\delta}{|S|} + (|S| - 1) \cdot \frac{-\delta}{|S|} = \frac{\delta n}{|S|} - \delta$.

Recall that the new center of subdomain π_c is the node with the highest load of the considered multiple-source FOS/C procedure. In other words, the node v is chosen such that $[w]_v$ is maximal over all $v \in V$. From the above it also follows that $[w]_v^S = \sum_{i \in S} [w]_v^i$. Due to Proposition 3.15 it holds that $\sum_{i \in S} [w]_v^i = \sum_{i \in S} [w]_i^v$. Thus, the new center z_c is the node for which the most load remains within the subdomain π_c in a single-source FOS/C procedure. Consequently, the contribution $\sum_{v \in \pi_c} [w]_v^{z_c}$ of this subdomain to F is maximized. As the maximization property holds for all subdomains, the claim follows. □

This result shows that the operation `ComputeCenters` determines an optimal set of centers *without* an exhaustive search among all nodes. Insofar it is also of interest for related methods, for example the k-means variant using Euclidean commute times (ECTD) as a distance measure [Yen 05]. On vertex-transitive graphs it computes the same solutions as Bubble-FOS/C due to Theorem 3.23. Let us therefore consider graphs that are not vertex-transitive. ECTD-k-means determines a new center v as $\arg\min_{v \in \pi_c} \sum_{u \in \pi_c} C[u, v]$. No explicit procedure that would be faster than brute force is provided by its authors. In the best case, which assumes that the subdomains are of asymptotically equal size n/k, the brute force running time is $\mathcal{O}(n^2/k)$ to compute all k center nodes, even if the pseudoinverse $\mathbf{L}^\dagger$ is known. The reason is that the distance of each node pair in a cluster needs to be considered; there are $\mathcal{O}(n^2/k^2)$ such pairs per cluster. Due to the similarities between FOS/C and ECTD established in Section 3.2 a faster way is possible, provided that the entries of $\mathbf{L}^\dagger$ can be accessed in constant time

and k is not too large:

$$\begin{aligned}\arg\min_{v\in\pi_c}\sum_{u\in\pi_c} C[u,v] &= \arg\min_{v\in\pi_c}\sum_{u\in\pi_c} l^\dagger_{v,v} + l^\dagger_{u,u} - 2l^\dagger_{u,v} &&= \arg\min_{v\in\pi_c}\sum_{u\in\pi_c} l^\dagger_{v,v} - 2l^\dagger_{u,v}\\ &= \arg\min_{v\in\pi_c} |\pi_c|\cdot l^\dagger_{v,v} - 2\sum_{u\in\pi_c} l^\dagger_{u,v} &&= \arg\min_{v\in\pi_c} |\pi_c|\cdot [w]^v_v - 2\sum_{u\in\pi_c}[w]^u_v\,.\end{aligned}$$

As shown in the proof of Lemma 4.4, the value of the sum $\sum_{u\in\pi_c}[w]^u_v$ for every node v can be computed by one multiple-source FOS/C procedure with source set π_c. In this way, the complexity of computing all centers is $\mathcal{O}(k\hat{c})$, where $\hat{c}$ denotes again the cost for solving one FOS/C procedure. In the case of an optimal solver, we get $\mathcal{O}(k\hat{c}) = \mathcal{O}(kn)$, which improves on $\mathcal{O}(n^2/k)$ in the case of $k \in o(\sqrt{n})$.

For the final convergence proof regarding Bubble-FOS/C we need one more proposition concerning the correctness of the algorithm.

Proposition 4.5. *During the execution of* Bubble-FOS/C *there are exactly k different center nodes and exactly k subdomains in each iteration τ.*

Proof. Observe that "$\leq$" is obvious, so that it suffices to show that there are at least k different center nodes and subdomains in each iteration. For the initial placement of centers we can easily ensure that k disjoint center nodes are chosen. Moreover, we know due to Proposition 3.8 that the centers determined by `ComputeCenters` belong to their own subdomain and must be different.

In the remainder we show that `AssignPartition` keeps each center in its current subdomain. Consider two arbitrary, but distinct centers z_i and z_j. Due to Proposition 3.15 we know that $[w]^{z_j}_{z_i} = [w]^{z_i}_{z_j}$. As $[w]^{z_i}_{z_i} > [w]^{z_i}_{z_j}$ (Proposition 3.8), we obtain $[w]^{z_i}_{z_i} > [w]^{z_j}_{z_i}$. Therefore, all center nodes remain in their subdomain. □

Theorem 4.6. Bubble-FOS/C *converges and produces a k-way partition. This partition is a local optimum of the potential function F.*

Proof. As F is maximized in every iteration of Bubble-FOS/C, it is strictly increasing in every iteration, until no local improvements are possible. Hence, since F is bounded, Bubble-FOS/C must always converge to a local maximum of F, which contains k subdomains due to Proposition 4.5. □

If Bubble-FOS/C is used for partitioning within a multilevel hierarchy, it converges very quickly. Our experiments indicate that usually three to five iterations of the main loop are sufficient to reach convergence on each level. This behavior has both a positive and a negative side. On the one hand, it shows the effectiveness of integrating Bubble-FOS/C into a multilevel hierarchy. The initial choice on each hierarchy level seems to be not far away from a local optimum. On the other hand, however, the quality of such a local optimum can be far away from the globally best value. Since convergence happens so quickly, the actual search space appears to be very narrow. Hence, one can expect

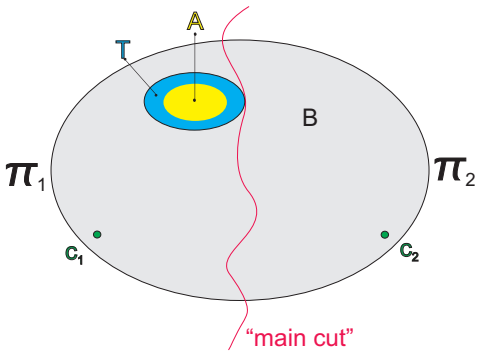


Figure 4.2.: Sketch of the situation assumed in Theorem 4.7.

that additional algorithmic techniques are necessary to escape bad local optima. Such a technique has been implemented for our related algorithm DibaP and is also viable for Bubble-FOS/C. Its idea is to choose the best among multiple solutions on a coarse level of the multilevel hierarchy (see Section 5.2.3). Other local search techniques might be profitable as well.

4.3.2. Connected Subdomains on Vertex-Transitive Graphs

For some applications that use partitioning or clustering as an intermediate step, it is important that all subdomains are connected, i. e., they have exactly one connected component each. Experiments with FEM graphs reveal that the subdomains computed by Bubble-FOS/C are (nearly always) connected if the algorithm is allowed to perform sufficiently many iterations. Unfortunately, we have not been able to verify this observation theoretically for all connected graphs. However, the following theorem makes a step towards this. It proves the connectedness of subdomains for all connected *vertex-transitive* graphs and $k = 2$ without requiring that Bubble-FOS/C has already converged. While the restriction to vertex-transitive graphs is mostly of theoretical interest, the result might become useful as a starting point for more general graph classes.

Theorem 4.7. *Let $G = (V, E)$ be a connected vertex-transitive graph. Fix two arbitrary different vertices $c_1, c_2 \in V$. Let the operation* `AssignPartition` *divide V into the two subdomains $\pi_1 = \{u \in V \mid [w]_u^{c_1} \geq [w]_u^{c_2}\}$ and $\pi_2 = \{u \in V \mid [w]_u^{c_1} < [w]_u^{c_2}\}$. Then, π_1 and π_2 are each connected components in G.*

Proof. First recall the following relationship between hitting times and the load vectors of FOS/C from Theorem 3.22:

$$[w]_u^v - [w]_v^v = \delta(H[v, v] - H[u, v])\,.$$

Also recall that $[w]_v^u = [w]_u^v$ holds for all $u, v \in V$ (Proposition 3.15) and that $H[v, v]$ is zero, which follows from the definition of hitting times. Furthermore, we can choose the drain constant δ to be one. From the proof of Theorem 3.23 we know that for all

vertex-transitive graphs $G = (V, E)$ and all $u, v \in V$ it holds that $[w]_u^u = [w]_v^v$, so that we obtain

$$[w]_v^v = [w]_u^u \wedge [w]_v^u = [w]_u^v \Rightarrow [w]_u^v - [w]_v^v = [w]_v^u - [w]_u^u \overset{\delta=1}{=} -H[u,v] = -H[v,u]\,.$$

Now assume for sake of contradiction that the subdomain π_2 is not connected. In this case there exists a node-separator $T \subseteq \pi_1$ such that there are at least two components $A, B \subseteq \pi_2$ which are not connected by a path via π_2. Assume w. l. o. g. that $c_2 \in B$, as shown in Figure 4.2. Then for some vertex $a \in A$ we obtain

$$\begin{aligned}
[w]_a^{c_2} > [w]_a^{c_1} &\Leftrightarrow [w]_a^{c_2} - [w]_{c_2}^{c_2} > [w]_a^{c_1} - [w]_{c_1}^{c_1} \\
&\Leftrightarrow H[c_2, c_2] - H[a, c_2] > H[c_1, c_1] - H[a, c_1] \\
&\Leftrightarrow H[a, c_1] > H[a, c_2]\,.
\end{aligned}$$

In the same manner we have for each vertex $x \in T$ that

$$H[x, c_1] \leq H[x, c_2]\,.$$

Let X_t be the random variable representing the node visited in timestep t by a random walk, and let $\mathcal{F}_u(x)$ be the event that a fixed vertex x is the first vertex visited in T of a random walk starting from some $u \in V$. Furthermore, denote by $\tau_a(T) := \min_{t\in\mathbb{N}}\{X_t \in T \mid X_0 = a\}$ and let $\tau_{a,T}(c_1) := \min_{t\in\mathbb{N}}\{X_t = c_1 \mid X_0 = a\} - \tau_a(T)$. By using conditional expectations ($\mathbb{E}[Y] = \sum_x \mathbf{Pr}[X = x]\,\mathbb{E}[Y \mid X = x]$) [Grim 01, p. 67], we obtain

$$\begin{aligned}
H[a, c_1] &= \mathbb{E}[\tau_a(c_1)] \\
&= \mathbb{E}[\tau_a(T) + \tau_{a,T}(c_1)] \\
&= \sum_{x\in T} \mathbf{Pr}[\mathcal{F}_a(x)] \cdot \big(\mathbb{E}[\tau_a(T) + \tau_{a,T}(c_1) \mid \mathcal{F}_a(x)]\big)\,,
\end{aligned}$$

which can be transformed by using the linearity of conditional expectations into

$$\begin{aligned}
&= \sum_{x\in T} \mathbf{Pr}[\mathcal{F}_a(x)] \cdot \big(\mathbb{E}[\tau_a(T) \mid \mathcal{F}_a(x)] + \mathbb{E}[\tau_{a,T}(c_1) \mid \mathcal{F}_a(x)]\big) \\
&= \sum_{x\in T} \mathbf{Pr}[\mathcal{F}_a(x)] \cdot \big(\mathbb{E}[\tau_a(x) \mid \mathcal{F}_a(x)] + \mathbb{E}[\tau_x(c_1) \mid \mathcal{F}_a(x)]\big) \\
&= \sum_{x\in T} \mathbf{Pr}[\mathcal{F}_a(x)] \cdot \big(\mathbb{E}[\tau_a(x) \mid \mathcal{F}_a(x)] + H[x, c_1]\big)\,.
\end{aligned}$$

Exactly the same arguments yield

$$H[a, c_2] = \sum_{x\in T} \mathbf{Pr}[\mathcal{F}_a(x)] \cdot \big(\mathbb{E}[\tau_a(x) \mid \mathcal{F}_a(x)] + H[x, c_2]\big)\,.$$

Due to $H[x, c_1] \leq H[x, c_2]$ for each $x \in T$ we finally obtain

$$\begin{aligned} H[a, c_1] &= \sum_{x \in T} \mathbf{Pr}\left[\mathcal{F}_a(x)\right] \cdot \left(\mathbb{E}\left[\tau_a(x) \mid \mathcal{F}_a(x)\right] + H[x, c_1]\right) \\ &\leq \sum_{x \in T} \mathbf{Pr}\left[\mathcal{F}_a(x)\right] \cdot \left(\mathbb{E}\left[\tau_a(x) \mid \mathcal{F}_a(x)\right] + H[x, c_2]\right) \\ &= H[a, c_2]\,, \end{aligned}$$

which is a contradiction to our assumption $H[a, c_1] > H[a, c_2]$. Therefore, the subdomain π_2 has to be connected.

The proof that π_1 is always connected is done in the same way. Assume the converse and let A and B be two disconnected components of π_1 with a node separator $T \subseteq \pi_2$ such that $c_1 \in B$. For a vertex $a \in A$ we have $H[a, c_1] \leq H[a, c_2]$ and for every vertex $x \in T$ it holds that $H[x, c_1] > H[x, c_2]$. Consequently,

$$\begin{aligned} H[a, c_2] &= \sum_{x \in T} \mathbf{Pr}\left[\mathcal{F}_a(x)\right] \cdot \left(\mathbb{E}\left[\tau_a(x) \mid \mathcal{F}_a(x)\right] + H[x, c_2]\right) \\ &< \sum_{x \in T} \mathbf{Pr}\left[\mathcal{F}_a(x)\right] \cdot \left(\mathbb{E}\left[\tau_a(x) \mid \mathcal{F}_a(x)\right] + H[x, c_1]\right) \\ &= H[a, c_1]\,, \end{aligned}$$

which is a contradiction to our assumption $H[a, c_1] \leq H[a, c_2]$, and the claim of the theorem follows. □

4.4. Algebraic Multigrid for Bubble-FOS/C

Most work performed by the Bubble-FOS/C algorithm consists in solving linear systems of the form $\mathbf{L}w = d$. More precisely, in each iteration of the main loop, $2k$ of these systems need to be solved. Since we assume that the input graphs are sparse ($m = \mathcal{O}(n)$), direct solvers are not applicable. They would cause a prohibitive running time (cubic) and memory consumption (quadratic). Instead, the solution process can be performed by the very popular Conjugate Gradient algorithm (CG) algorithm, which is suitable for symmetric positive semidefinite systems as long as the right-hand side is consistent [Kaas 88], i. e., as long as a solution exists. Yet, the convergence of CG tends to slow down considerably when the linear systems stemming from numerical simulations become larger [Shew 94]. Furthermore, recall that within one Bubble-FOS/C operation only the drain vector d differs for each subdomain since the matrix $\mathbf{L}$ depends only on the graph and is the same for all k FOS/C procedures.

4.4.1. Fundamentals of Algebraic Multigrid

The decreasing convergence speed of CG and the multiple occurrence of $\mathbf{L}$ are exploited in the following by applying an algebraic multigrid (AMG) solver. Multigrid methods

(e. g., [Trot 00]) are among the fastest iterative solvers and preconditioners for large linear systems derived from a wide class of partial differential equations. They are based on the observation that relaxation methods such as Jacobi or Gauss-Seidel eliminate high-frequency (unsmooth) error components in the solution vector very effectively. However, the reduction of low-frequency error components takes them a very large number of iterations. That is why these relaxation methods are also called *smoothers*.

In order to eliminate also the low-frequency error quickly, a multigrid algorithm uses a hierarchy of matrices (also called *grids*, leading to the name *multigrid*), whose size decreases from one hierarchy level to the next one. If one passes a linear system with smooth error to the next coarser matrix of the hierarchy, the low-frequency components become oscillatory (unsmooth) again. Consequently, these oscillatory error components can be smoothed efficiently by relaxation methods again. Similar to the multilevel paradigm used for graph partitioning, this process is continued recursively, until the linear system on the coarsest level is small enough to be solved directly with adequate resource consumption.

AMG is an extension of classical multigrid to cases where no geometric information is available in connection with the matrix. One of the major differences between the two methods is the construction of the hierarchy. Classical geometric multigrid methods operate on fixed hierarchies, which sometimes have very simple construction rules. These hierarchies can also be derived from successive mesh refinements performed by the meshing algorithm of the underlying numerical application.

In contrast to this, AMG constructs its own hierarchy by a top-down coarsening approach. For this, only the matrix corresponding to the finest mesh is necessary. That is why AMG, as opposed to geometric multigrid, is sometimes seen as a "black box approach". On the other hand, such a categorization is only half the truth. To be highly effective, AMG requires for non-standard problems a wise choice of components as well as parameters. For positive semidefinite matrices such as the Laplacian matrices arising in FOS/C procedures, this guidance by the user or developer is also necessary. Nevertheless, AMG is so appealing to us for two reasons. First, a successful tuning can be expected to result in a very efficient solver. Second, its hierarchy construction has to be made only once and can then be reused for all linear systems with the same matrix $\mathbf{L}$. Hence, this sometimes expensive task can be amortized over at least $2k$ systems in our application.

4.4.2. General AMG Coarsening and Solution Process

Coarsening a matrix $\mathbf{L} = \mathbf{L_f}$ to obtain the coarse matrix $\mathbf{L_c}$ (f = fine, c = coarse) of the next hierarchy level consists of three main steps: First, one determines the coarse vertices that are transferred to the next level. They must facilitate a significant reduction of the number of nodes and edges in the next level. Moreover, they have to be able to interpolate those nodes accurately which are not retained within the coarse matrix. That

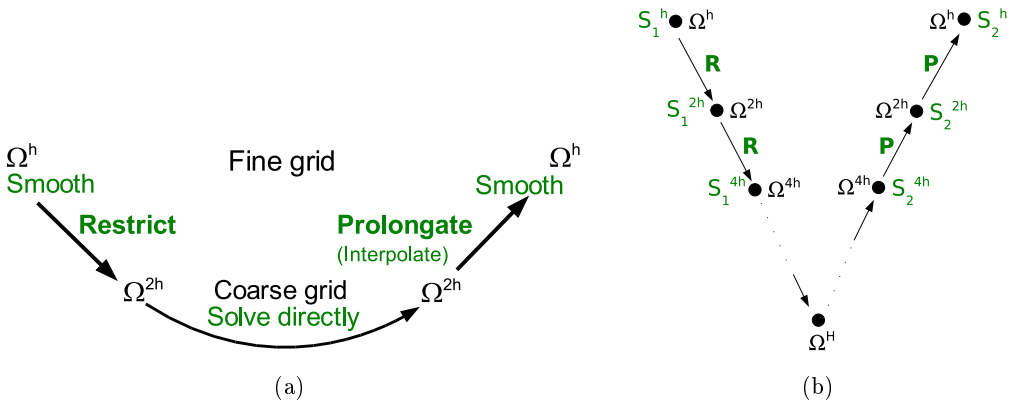


Figure 4.3.: (a) Two-grid cycle with coarse grid correction. (b) Sketch of V-cycle.

is why one prefers nodes with many strong couplings. In graph terminology these are nodes which are connected to many other nodes by edges with a high weight (see Briggs et al. [Brig 00, Ch. 8] for more details).

After having computed the set of coarse nodes, one determines interpolation weights and sets up the *prolongation* matrix $\mathbf{P}$. The choice of the interpolation weights should be consistent with the problem and the coarsening scheme employed since it is crucial for a fast convergence of the solver [Stub 01]. The coarse matrix is determined by applying the *Galerkin principle*, i. e., $\mathbf{L_c}$ is computed as $\mathbf{L_c} := \mathbf{R}\mathbf{L_f}\mathbf{P}$ ($\mathbf{R} = \mathbf{P}^T$ is the *restriction* matrix), the Galerkin operator. The weight of an entry (u, v) of the coarse matrix $\mathbf{L}_c$ is therefore computed as $(\mathbf{L}_c)_{u,v} = \sum_{i=1}^{n}\sum_{j=1}^{n} r_{u,i}l_{i,j}p_{j,v} = \sum_{i=1}^{n}\sum_{j=1}^{n} p_{i,u}l_{i,j}p_{j,v}$. Hence, the edge weights of coarse nodes are computed as weighted aggregations. Each aggregation considers nodes of distance at most 3 in the fine matrix/graph. This process of obtaining a coarser representation of the original matrix can be continued recursively to yield a complete multigrid hierarchy.

Following the hierarchy construction in the setup phase, the actual solution process is performed by an algorithm which consists of the following main operations: presmoothing, restriction, solving the coarse problem recursively, interpolating the coarse solution, and postsmoothing. This is shown for two matrices in Figure 4.3(a), where Ω^h and Ω^{2h} denote the matrices corresponding to the fine and the coarse discretization domain, respectively. Using recursion, such a scheme can be applied to the complete hierarchy. Depending on the way the recursive traversal is performed, different solution algorithms (so-called *cycles*) are distinguished. For example, due to its shape, going down the hierarchy and up again to the finest level is called a *V-cycle* (see Figure 4.3(b); Ω^H denotes the coarsest matrix, S_1 and S_2 smoothing operators).

4.4.3. Details of our AMG Implementation

After the description of the general ingredients of an AMG algorithm, we explain in this section how they are realized in our implementation for solving FOS/C procedures. First

of all, we show that AMG is actually suitable for solving the occurring linear systems. This is not self-evident since the traditional multigrid theory has been developed for nonsingular matrices [Stub 00]. If no virtual vertex is used, the Laplacian matrices in our FOS/C procedures do not fall into this category. Since AMG is often faster than other iterative methods, there have been attempts to transfer the algorithm to other graph classes as well, including singular matrices (e. g., [Virn 07]). Neglecting numerical issues, we can show that AMG is applicable to our problem of solving FOS/C procedures with symmetric positive semidefinite Laplacian matrices as well:

Proposition 4.8. *An FOS/C procedure represented by a linear system of the form* $\mathbf{L}w = d$ *can be solved by an AMG algorithm.*

Proof. There exists a solution for the equation $\mathbf{L}w = d$ because $d \perp \mathbf{1}$ (Theorem 3.3). Now consider a two-level procedure. Let the matrix and the vectors of the fine level be subscripted with f (for fine), those of the coarse level by c (for coarse). In particular: $\mathbf{L}_f := \mathbf{L}$, $w_f := w$, and $d_f := d$.

Let the approximation of the solution vector w be denoted by $\hat{w}$. The residual vector r_f is defined as $r_f := d_f - \mathbf{L_f}\hat{w}_f$. It holds that $r_f \perp \mathbf{1}$ since $d_f \perp \mathbf{1}$ and $\mathbf{L_f}\hat{w}_f \perp \mathbf{1}$. The latter can be verified easily as follows: $\langle \mathbf{L}_f\hat{w}_f, \mathbf{1}\rangle = \sum_{i=1}^{n}\sum_{j=1}^{n} l_{i,j}\hat{w}_j = \sum_{j=1}^{n}\hat{w}_j\sum_{i=1}^{n} l_{i,j} = \sum_{j=1}^{n}\hat{w}_j \cdot 0 = 0$.

On the coarse level we need to solve $\mathbf{L_c}e_c = r_c$ for the coarse error e_c, where $\mathbf{L_c} = \mathbf{R}\mathbf{L_f}\mathbf{P}$ and $r_c = \mathbf{R}r_f$. Since the prolongation matrix has been constructed such that the column sum of $\mathbf{R}$ (or, equivalently, the row sum of $\mathbf{P}$) is 1, $r_c \perp \mathbf{1}$ holds as well. Moreover, $\mathbf{L_c}$ is again symmetric positive semidefinite. Symmetry follows from: $[\mathbf{L}_c]_{u,v} = [\mathbf{R}\mathbf{L_f}\mathbf{P}]_{u,v} = \sum_{i=1}^{n}\sum_{j=1}^{n} r_{u,i}l_{i,j}p_{j,v} = \sum_{j=1}^{n}\sum_{i=1}^{n} r_{v,j}l_{j,i}p_{i,u} = [\mathbf{L}_c]_{v,u}$. One can also verify for an arbitrary vector x of compatible size: $\langle \mathbf{L}_c x, x\rangle = \langle \mathbf{R}\mathbf{L_f}\mathbf{P}x, x\rangle = \langle \mathbf{L_f}\mathbf{P}x, \mathbf{R}^T x\rangle = \langle \mathbf{L_f}\mathbf{P}x, \mathbf{P}x\rangle$. For $y := \mathbf{P}x$ follows $\langle \mathbf{L_f}y, y\rangle \geq 0$ because $\mathbf{L}_f$ is positive semidefinite and therefore also $\mathbf{L}_c$. Consequently, the coarse problem has a solution, too. This argument can be continued recursively, so that an algebraic multigrid method is applicable. □

While Proposition 4.8 helps to estimate if AMG can be useful at all for us, it does not consider the actual convergence behavior. The latter has been investigated by Friedhoff and Heming [Frie 07]. They state conditions under which an AMG algorithm can be proved to converge while solving FOS/C procedures.

As the next step, we describe the two coarsening algorithms used in our implementation. Then, after giving details on the applied interpolation scheme, we describe briefly which solution algorithms have been implemented and what their differences are.

4.4.3.1. Selecting Coarse Nodes

The mechanism for selecting the nodes that are retained in the coarse matrix has to fulfill several properties to meet our requirements. First of all, it needs to ensure a fast convergence. Another objective, which conflicts with fast convergence, is a fast

reduction of the matrix size. Smaller matrices lead to less memory consumption and a faster execution time per iteration in the solution phase. On the other hand, a more aggressive coarsening can affect the convergence behavior in a negative way. That is why one needs to find a good balance between these two goals. For our purposes a relatively aggressive scheme is preferred in order to keep the number of hierarchy levels small. As we will see later on, this is important for fast multilevel partitioning. Moreover, if AMG is used for matrices describing 3D problems, mild coarsening tends to increase the memory consumption to an undesirable amount [Ster 06].

That is why we use PMIS coarsening [Ster 06], which has been developed specifically for keeping the memory requirements moderate in 3D problems. It also keeps the number of created hierarchy levels rather small. The selection of the coarse nodes by PMIS uses the notion of maximum weight independent sets (MWIS) from graph theory. All nodes $v \in V$ get a weight g_v, which denotes the number of strong couplings to the neighbors of v. (Recall that, generally speaking, strong couplings are off-diagonal entries whose absolute value is above a threshold.) Then, based on these weights, PMIS computes an MWIS $\mathcal{I}$, which becomes the (preliminary) set of coarse nodes C. That way, two properties are achieved. The set C is very small and nodes are preferred that can interpolate many other nodes accurately because they have many strong couplings. However, MWIS is an $\mathcal{NP}$-hard problem, so that we employ a fast greedy heuristic, which looks for nodes with locally highest weight. In addition, nodes not in $\mathcal{I}$ that have no strongly coupled neighbor in C have to be moved from the set of non-coarse (or fine) nodes, which is denoted by $F := V \backslash C$, to C.

In cases where PMIS coarsens too much, we neglect its result. Instead, CLJP coarsening [Hens 02] is applied, which tends to coarsen less aggressively. The major algorithmic difference between the two is that CLJP adjusts the node weights g_v during the process. More precisely, it reduces the weight of vertices whose neighbors are inserted into C by $q := 1$. This adjustment reflects that the coupling to a coarse neighbor has already been taken into account. In our implementation we also vary the size of q adaptively. This gives us more control over the resulting matrix size.

4.4.3.2. Interpolation Weights

Our experiments with different interpolation schemes confirm general experience [Stub 01] that this choice is crucial in order to obtain a satisfying convergence of the solver. A simple M-matrix interpolation [Stub 00, p. 448] leads to a very slow convergence in connection with the coarsening schemes used, when our FOS/C procedures have more than approximately 50,000 nodes.

Our second choice, called *classical interpolation*, works well for our class of problems. It has also been used by Safro et al. [Safr 06] within a multilevel approach for optimizing linear orderings of matrices. For variables (resp. nodes) i, j of the current matrix (resp. graph) let $I(j)$ denote the index of node j in the coarse matrix and define N_i as the

Algorithm 3 FMV-CYCLE($\mathbf{L_f}$, w, d, `level`)$\rightarrow w$

```
   /* Precondition: FMV-CYCLE and V-CYCLE have access to */
   /* the matrix hierarchies of L, R, and P, */
1  if L_f is coarse enough then
2    w = V-CYCLE(L_f, w, d, level);
3  else
     /* Restriction of residual: */
4    r = R(d − L_f w);
     /* Recursive call with L_c = L[level+1]: */
5    e = FMV-CYCLE(L_c, e, r, level + 1);
     /* Interpolation of coarse error: */
6    w = w + Pe;
     /* Call to V-cycle: */
7    w = V-CYCLE(L_f, w, d, level);
8  return w;
```

neighbors i that are in the coarse set C.

$$[\mathbf{P}]_{i,I(j)} = \begin{cases} \omega_{i,j} / \sum_{k \in N_i} \omega_{i,k} & \text{for } i \in F,\, j \in N_i\,, \\ 1 & \text{for } i \in C,\, j = i\,, \\ 0 & \text{otherwise}\,. \end{cases}$$

Weights below a given threshold η (e. g., $\eta = 1/16$) are not included. This truncation reduces the number of nonzero entries in the coarse matrix. Consequently, it saves memory space and solution time. If the threshold is not too large (one even finds $\eta = 1/5$ in the literature [Safr 06]), the convergence speed is hardly affected. After a truncation the remaining values of a row are scaled such that the row sum in $\mathbf{P}$ is always 1. In this way, as also pointed out by Safro et al., one can interpret the entry $[\mathbf{P}]_{i,I(j)}$ as the likelihood of i to belong to the aggregate at position $I(j)$.

4.4.3.3. Solution Phase

For the solution phase we have implemented two algorithms, the V-cycle (Figure 4.3 (b)) and the full multigrid V-cycle (FMV-cycle) (cf. [Brig 00, Ch. 3] or Algorithm 3). Both algorithms have access to the hierarchy constructed in the setup phase. The V-cycle algorithm can act as an iterative solution algorithm itself. Alternatively, a V-cycle can be used within an FMV-cycle. All calls to V-cycles made by an FMV-cycle can be iterated more than once, but in our implementation only one iteration is used. A standard CG implementation serves us as the direct solver on the lowest level.

To solve a linear system $\mathbf{L}w = d$ iteratively, both algorithms are repeated, until the desired error tolerance in the residual is reached. One FMV-cycle is more costly than one V-cycle, but its convergence is also better. Our experiments show clearly that, in

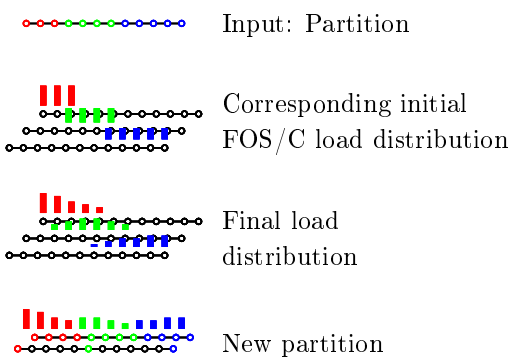


Figure 4.4.: Schematic view of the `Consolidation` operation.

total, the gain in convergence speed outweighs the increased costs per cycle. Hence, we use the faster FMV-cycle as AMG solution algorithm in all our subsequent experiments.

4.5. Extensions to Bubble-FOS/C for Graph Partitioning

4.5.1. `Consolidation`: Mixing `AssignSubdomain` and `ComputeCenters`

Recall that the loop in Lloyd's k-means algorithm consists only of two alternating operations. For our purposes they have been transformed into `AssignSubdomain` and `ComputeCenters`. In case Bubble-FOS/C is used for graph (re)partitioning, we also use another operation, which is a mixture of the two. It is called `Consolidation` and computes a new partition from a given one. Its process is sketched in Figure 4.4. The initial load distribution is the same as for `ComputeCenters`. Hence, the source set contains all nodes of the subdomain currently considered. Yet, after computing the corresponding FOS/C procedure for each subdomain, the resulting load distributions are evaluated as in `AssignSubdomain`. Each node is assigned to the subdomain from which it has received the highest load. In summary, `Consolidation` computes the FOS/C convergence load of `ComputeCenters`, but identifies new subdomains instead of centers from that.

Our experiments with the `Consolidation` operation indicate its potential for graph partitioning. Due to the different sizes of the source sets, it contains an implicit balancing method. This results from a steeper load distribution obtained with smaller source sets. Usually this balancing is not sufficient to obtain almost equally sized subdomains. In particular because `Consolidation` works only reasonably well if the subdomain sizes do not differ extremely. Yet, it is a diffusion-based process that has some balancing capabilities, moves subdomains in the direction of their desired positions, and can be integrated easily into Bubble.

The balancing property makes `Consolidation` unsuitable whenever Bubble-FOS/C is used for graph clustering. There, equally sized subdomains are rarely the correct solution. Apart from these practical considerations, there is also a theoretical drawback of using `Consolidation`. So far, it is unclear how to transfer the convergence proof of Bubble-FOS/C to the extended algorithm with `Consolidation` operations. This is

mainly due to the fact that, unlike the potential function F, `Consolidation` does not use the notion of subdomain centers.

4.5.2. Balancing Methods

When Bubble-FOS/C is used for graph (re)partitioning, we need to ensure that it generates partitions which stay within the user-defined imbalance. That is why two additional balancing operations are employed. Since large parts of them have been developed by Schamberger [Scha 06, p. 87ff.] and our modifications affect more the implementation than the algorithmic idea, we describe the operations only briefly.

The first one takes the k FOS/C load vectors computed by an `AssignPartition` or `Consolidation` operation. Then, it determines for each vector w_c, $c \in \{1, \ldots, k\}$, a scalar value ξ_c with the following property. If all w_c are multiplied with their respective ξ_c, a new evaluation of the resulting load values yields a (more) balanced partition. This procedure called ScaleBalance is relatively simple and fast. Moreover, it retains the good properties (like the shape) of the subdomains. The balancing succeeds in many cases, but there are exceptions to this rule. These include situations with a very high imbalance or very large k.

Whenever balancing by scaling the load vectors is not successful, an additional balancing method called FlowBalance is employed. First, it computes the amount of nodes that have to migrate between different subdomains. This computation is done by diffusive load balancing to achieve ℓ_2-minimal migration costs [Diek 99]. Then, for each node v and each subdomain c, a priority p_v^c is computed as

$$p_v^c := \frac{[w]_v^c}{[w]_v^{\Pi(v)}} \,.$$

Similar to the priority values for boundary smoothing (Section 4.1.2), the priorities p_v^c computed here express how certain the affiliation of a node $v \in V$ to its current subdomain is. (A value close to one indicates a low certainty because the respective node is also drawn to another subdomain.) The nodes are then moved in the order of these priorities, so that the "uncertain" nodes are migrated first. These moves are repeated, until the balancing flow is saturated and the partition Π balanced.

A priority-based order of node moves results in sequential parts of this process. One could argue that the number of computations is small compared to solving many linear systems. Yet, if executed in parallel on different processors, it would require many communication steps to ensure the correct order of migration steps. That is why we have also developed a different version more suitable for parallel computers with distributed memory. The original idea is modified such that the movement of nodes is performed in rounds, where each round moves the nodes that are at the current subdomain borders. Additional techniques avoid moves to interfere with each other. That way, the number of communication operations for updating priorities can be reduced.

Algorithm 4 BUBBLE-FOS/C-PART(G, k, Π, `maxOuter`, `maxInner`) $\rightarrow \Pi$

```
01  if Π is defined
02  then /* ComputeCenters */
03    parallel for c = 1,...,k do
04      Initialize d_c (S = π_c)
05      Solve L w_c = d_c
06      z_c = argmax_{v∈π_c} [w_c]_v
07  else /* Find initial centers */
08    Z = LoadBased-
        InitialCenters(G, k)
    /* Initial AssignSubdomain */
09  parallel for c = 1,...,k do
10    Initialize d_c (S = {z_c})
11    Solve and normalize L w_c = d_c
12  parallel for each v ∈ V do
13    Π(v) = argmax_{c∈{1,...,k}} [w_c]_v
14  for τ = 1 to maxOuter
      /* ComputeCenters */
15    parallel for c = 1,...,k do
16      Initialize d_c (S = π_c)
17      Solve L w_c = d_c
18      z_c = argmax_{v∈π_c} [w_c]_v
      /* AssignSubdomain */
19    parallel for c = 1,...,k do
20      Initialize d_c (S = {z_c})
21      Solve and normalize L w_c = d_c
22    parallel for each v ∈ V do
23      Π(v) = argmax_{c∈{1,...,k}} [w_c]_v
24    Π = ScaleBalance(G, k, Π, W)
25    for j = 1 to maxInner
        /* Consolidation */
26      parallel for c = 1,...,k do
27        Initialize d_c (S = π_c)
28        Solve and normalize L w_c = d_c
29      parallel for each v ∈ V do
30        Π(v) = argmax_{c∈{1,...,k}} [w_c]_v
31      Π = ScaleBalance(G, k, Π, W)
32    Π = FlowBalance(G, k, Π, W)
33  return Smooth(Π)
```

4.5.3. The Extended Algorithm

The extended BUBBLE-FOS/C algorithm is depicted as Algorithm 4. It includes the aforementioned features `Consolidation` and balancing. This integration requires some changes in the order of the operations, but no modifications of the underlying ideas. Note that W denotes the $n \times k$ matrix whose columns are the k load vectors w_1 to w_k.

Implemented in this way, BUBBLE-FOS/C is suitable for partitioning graphs into equally sized subdomains. To consider other practical aspects, additional adaptations have been made. For example, in practice one does not wait until the main loop converges. Instead, the loop is stopped after a fixed number of iterations. Using the multilevel paradigm, a very small number of iterations is already sufficient to obtain good solutions. This paradigm and our modifications to the standard multilevel approach are described next.

4.6. Multilevel Paradigm with Algebraic Multigrid

For a good performance in terms of speed and quality of the BUBBLE-FOS/C algorithm, it is crucial to bring about its convergence to a locally optimal solution after a constant number of iterations. This can be achieved by employing a multilevel scheme. The multilevel concept is known to work well for graph partitioning [Hend 95a] and has also been

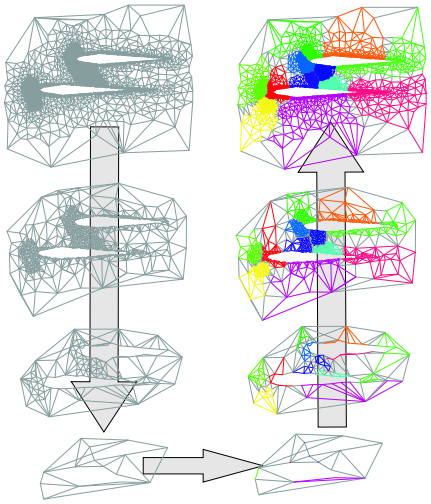


Figure 4.5.: Illustration of multilevel paradigm.

used for graph clustering [Dhil 07]. Its idea is illustrated in Figure 4.5 and has already been described in Section 1.3.1. In summary, the main steps are recursive coarsening (left side of Figure 4.5), computing an initial solution on the coarsest graph (bottom), and then recursive interpolation and local improvement (right side). The underlying idea is that on each level but the coarsest, the local improvement process is started with an already reasonable solution. Hence, the convergence of Bubble-FOS/C can be expected to be very fast, so that the additional work is more than compensated.

Most state-of-the-art partitioning libraries use approximate maximum weight matchings or very similar methods for coarsening within the multilevel scheme. For Bubble-FOS/C we opt for an alternative way. Recall that linear systems have to be solved for Bubble-FOS/C, which is done by algebraic multigrid (AMG). As we have explained above, AMG also constructs a hierarchy of graphs. Hence, instead of computing an additional hierarchy based on matchings, we use the existing AMG hierarchy. That is why a fast reduction of the matrix sizes is important in our AMG coarsening schemes. Deep hierarchies would cause long running times, possibly even without yielding much better solutions than more shallow hierarchies.

Similar ideas of using AMG for providing multilevel hierarchies have been pursued independently by Safro et al. [Safr 06]. In a simpler form it has been used for a multilevel procedure computing the Fiedler vector z_2 for spectral partitioning [Drie 95].

Note that for the local improvement on any given hierarchy level l, the AMG algorithm within Bubble-FOS/C needs to start on level l, too. However, if AMG were called as a black-box solver on level l, it would construct a completely new hierarchy with level l as its top. That is why we have adapted the standard AMG algorithm such that no duplicate hierarchies are built. If started on level l, all solution algorithms in our implementation use the hierarchy that has been built at the beginning with the finest

graph/matrix as its topmost level.

4.7. Experimental Results

The experiments are based on our C/C++ implementation of the extended Bubble-FOS/C algorithm for graph partitioning (Algorithm 4) and the generic Bubble-FOS/C algorithm for graph clustering (Algorithm 1). They have been conducted on a computer equipped with an Intel Core 2 Duo 6600 CPU and 1 GB RAM. The operating system is Linux (openSUSE 10.2, Kernel 2.6.18) and the main code has been compiled with the Intel C/C++ compiler 10.0. The compiler uses level 2 optimization and auto-parallelization for all programs under consideration (except for Graclus, which has been compiled with GCC 4.1). For Bubble-FOS/C we also use POSIX threads for parallelizing some parts of the hierarchy construction and the solution of linear systems. This allows for the use of both processor cores. Threads are not available for the serial libraries METIS, Jostle, and Graclus, which serve as standards of reference for graph partitioning (METIS and Jostle) and graph clustering (Graclus). This aspect is discussed in more detail in Section 4.7.4.

An issue to consider is the dependence on "random influences". As an example, the order in which the vertices are stored within the graph data structure play a significant role for node-exchanging partitioning algorithms like KL/FM [Scha 03, Elsn 05] (and also kernel k-means with local search, as in Graclus). This is due to the fact that the order in which the nodes are inserted into the gain buckets determine the order of the corresponding moves. Different orders can lead to different local optima. Hence, in our experiments the programs METIS, Jostle, and Graclus are run ten times on the same graph, but with a randomly permuted vertex set. The edges are permuted accordingly, so that the resulting graphs are isomorphic to the original one. For Bubble-FOS/C the order of the vertices is insignificant because the diffusive partitioning operations are hardly affected by it. Only in the rare case of ties in the load values small changes can occur, which are mostly irrelevant. That is why we account for random influences in Bubble-FOS/C by performing ten runs on the same graph with different random seeds, resulting in different choices for the first center vertex.

4.7.1. Bubble-FOS/C on the 2D Torus

Grid graphs are frequently used as finite element discretizations of planar domains. Often, they are adaptively refined, so that different areas of the grid have a different number of elements per area. For simpler problems, however, also grids of the same resolution are used. It is also not uncommon to use cyclic boundary conditions, which are simulated by a torus instead of a grid. Insofar it is interesting to see how partitions computed by Bubble-FOS/C on a 2D torus look like and which properties they have.

For an assessment of `AssignPartition` recall important properties of FOS/C on the

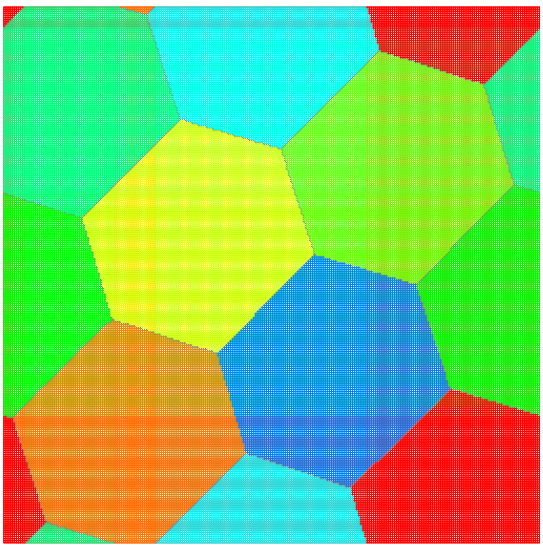


Figure 4.6.: Partition with eight subdomains of a 256×256 torus computed by Bubble-FOS/C.

torus. Its convergence load is symmetric (Proposition 3.15) and monotonous (Theorem 3.46). Moreover, all single-source FOS/C procedures yield the same convergence load vector up to permutation. Recall that we can conclude from Mangad's result (Theorem 3.47) that, asymptotically, FOS/C yields circular isolines on an infinite 2D-grid. Moreover, based on our experimental data and theoretical results of Ellis [Elli 01a], we also know for the torus that the isolines of the load have a certain shape, which is close to circular in a not too far distance from the source node. In many cases the subdomain boundaries tend to be in such a distance from their nearest source nodes where the isolines are circular. Hence, the subdomain affiliation is decided implicitly by geometry, namely by Euclidean distances corresponding to the radii of the isolines.

If we assume that certain degeneracies such as discretization errors can be neglected, the assignment of nodes to subdomains based on Euclidean distances results in subdomains with the *Voronoi property*. This means that each subdomain contains those nodes which are closest to its center node w. r. t. to Euclidean distance. Voronoi cells have the property to be connected and convex, which follows from the fact that they are intersections of halfplanes [Berg 97, Ch. 7].

The correspondence to Voronoi cells and their convexity is an indication that Bubble-FOS/C indeed computes partitions with short boundaries, at least on a torus. This indication is further confirmed experimentally. With a large enough number of iterations, Bubble-FOS/C computes the regular hexagonal tessellation of the torus depicted in Figure 4.6, even without balancing or smoothing. In the geometric setting such a partition is known to be the global optimum of Lloyd's geometric k-means algorithm [Newm 82] and to have the shortest possible boundary (e. g., [Puu 05]). Although being no rigorous proof, these observations give some evidence that Bubble-FOS/C performs well on torus graphs and those with some similarity to the torus or grid structure.

Table 4.1.: Graphs used in the experiments of Section 4.7.2.

	Size		Degree			
Graph	$\|V\|$	$\|E\|$	min	max	avg	Origin
airfoil1	4,253	12,289	3	9	5.779	FEM 2D
crack	10,240	30,380	3	9	5.934	FEM 2D
whitacker (dual)	19,190	28,581	2	3	2.979	FEM 2D dual
biplane9	21,701	42,038	2	4	3.874	FEM 2D
stufe10	24,010	46,414	2	4	3.866	FEM 2D
altr4	26,089	163,038	5	24	12.499	FEM 3D
shock9	36,476	71,290	2	4	3.909	FEM 2D
wing	62,032	121,544	2	4	3.919	FEM 3D dual

4.7.2. Graph Partitioning

4.7.2.1. Experimental Settings

For the experiments presented in this section we have chosen eight graphs of small to medium size, see Table 4.1, that are or have been frequently used as benchmark instances. This sample is on the one hand large enough to draw valid conclusions from its results, on the other hand it is small enough to keep the evaluation efforts of very detailed experiments tolerable and the presentation of the results concise. Note that, apart from the extensive experiments presented below, we have used more graphs than these eight ones in further experiments with Bubble-FOS/C and its competitors. These additional results confirm the general trend and are therefore omitted here.

The choice of the metrics used for comparing different programs or algorithms plays a major role for the evaluation, too. This choice is certainly based on the application for which the graph partitioners are employed. Since we focus on numerical simulations, we do not only consider the edge-cut (EC). As Hendrickson has pointed out [Hend 98], the number of boundary nodes (BN, *bnd*) is a more accurate measure for the communication within numerical solvers than the edge-cut. Note that the edge-cut is the summation norm of the external edges (*ext*) divided by 2 to account for counting each edge twice. For some applications not only the summation norm ℓ_1 of *ext* and *bnd* over all k partitions has to be considered, but also the maximum norm ℓ_∞. This is particularly the case for parallel simulations, where all processors have to wait for the one computing longest. That is why we record *ext* and *bnd* in both norms. Their formal definition is given in Section 2.1.1. If tables are used for presenting results, the best value in each category is usually written in bold font.

4.7.2.2. Finding suitable Loop Parameters

Our first objective is to determine suitable values for the number of loop iterations within Bubble-FOS/C. The iteration count of the outer loop (`maxOuter`), during which the assignment to subdomains and the center computation take place, is abbreviated by AC,

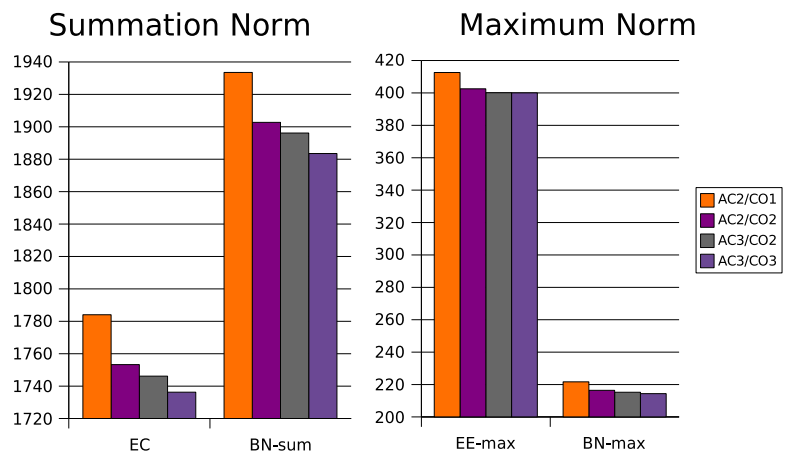


Figure 4.7.: Number of external edges and boundary nodes of BUBBLE-FOS/C solutions in different loop parameter settings AC/CO for ℓ_1- and ℓ_∞-norm, averaged over all k.

while the iteration count of the inner loop (`maxInner`) is abbreviated by *CO*, which stands for consolidation. In Figure 4.7 we compare the number of external edges and boundary nodes in the summation (left) and the maximum norm (right) for four different iteration number combinations. The results are averaged over all graphs in the benchmark set and all $k \in \{4, 8, 12, 16, 20\}$. A detailed presentation of the results for each k can be found in Tables A.1 and A.2 in the appendix. The average running times show no surprising behavior; they range from 7.87s for AC2/CO1 to 18.14s for AC3/CO3.

Regarding the quality, the combination of three outer and three inner loop iterations generally yields the best results. This is hardly surprising because this combination also invests the highest amount of running time in our experiments. Observe that, in particular in the maximum norm, the two other settings AC2/CO2 and AC3/CO2 are hardly worse than the best one. It seems that in many cases AC3/CO2 computes a partition close to a local optimum of BUBBLE-FOS/C. Thus, additional loop iterations do not yield significant improvements.

4.7.2.3. Influence of Linear Solver and Virtual Vertex

Our most significant algorithmic modification to previous BUBBLE-FOS/C implementations (the latter are based on our work with Schamberger [Meye 05, Meye 06c, Scha 06]) is the introduction of AMG for solving the linear systems within FOS/C procedures and for providing a multilevel hierarchy. To judge how running time and quality are affected by this modification, we compare it to our previous implementation of BUBBLE-FOS/C, which uses CG as linear solver and a multilevel coarsening based on matchings.

Figure 4.8 compares the solution quality achieved by both variants (with AMG and with CG) for the parameter combination AC3/CO2. The values shown in the respective two leftmost columns are the external edges (EC/EE) and boundary nodes (BN) in both

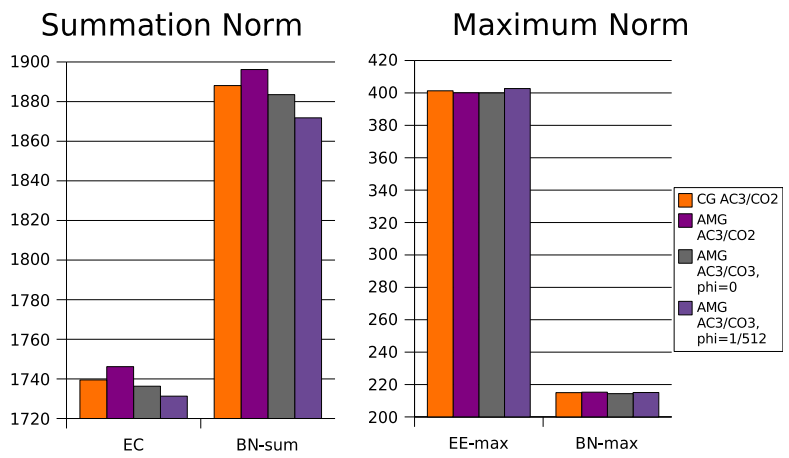


Figure 4.8.: Quality comparison (number of external edges and boundary nodes in both norms) between the CG and the AMG version of Bubble-FOS/C (two leftmost bars in each column) and between the use of FOS/C and FOS/V as similarity measure (two rightmost bars), averaged over all k.

relevant norms (ℓ_1 and ℓ_∞), averaged over all graphs in the benchmark set and over all k. A more detailed presentation of these data can be found in Table A.3 in the appendix. These results indicate that the solution quality of both Bubble-FOS/C variants are very similar, although they show a slight advantage to the CG solver combined with a multilevel matching hierarchy. A possible reason for this small discrepancy could be the different coarsening approaches of AMG and the matching algorithm. While the matching algorithm is adjusted such that star-like subgraphs (few nodes with high weights and many nodes with low weights) are avoided, AMG coarsening often produces these stars. Sometimes such subgraphs can be disadvantageous for multilevel partitioning [Moni 04] and future work could address this issue in connection with AMG. Yet, the loss in quality is well below 1% and therefore rather small. In comparison the running time improvement of the AMG version is significant. As shown in Figure 4.9 by the two topmost bars in each row, the running times are reduced by a factor between 4 and 5.

Also note that the speedup, whose detailed numerical values are displayed in Figure 4.10, between the two methods increases when k becomes larger. These data show that the introduction of AMG within Bubble-FOS/C constitutes a considerable acceleration for the benchmark graphs. Hence, as expected, the much more involved implementation of a multigrid solver – compared to the relatively simple CG – pays off. Since the convergence rate of the CG solver, unlike that of AMG, depends on the system size, one can expect that the speedup achieved by AMG increases for larger graphs. One has to consider, however, that solving large linear systems with AMG is not inexpensive, either. This stems from the fact that the hierarchy construction can become very costly because large matrices have to be multiplied with each other.

So far, both variants of Bubble-FOS/C using the different linear solvers are run without making use of the virtual vertex notion described in Section 3.5. It is also of

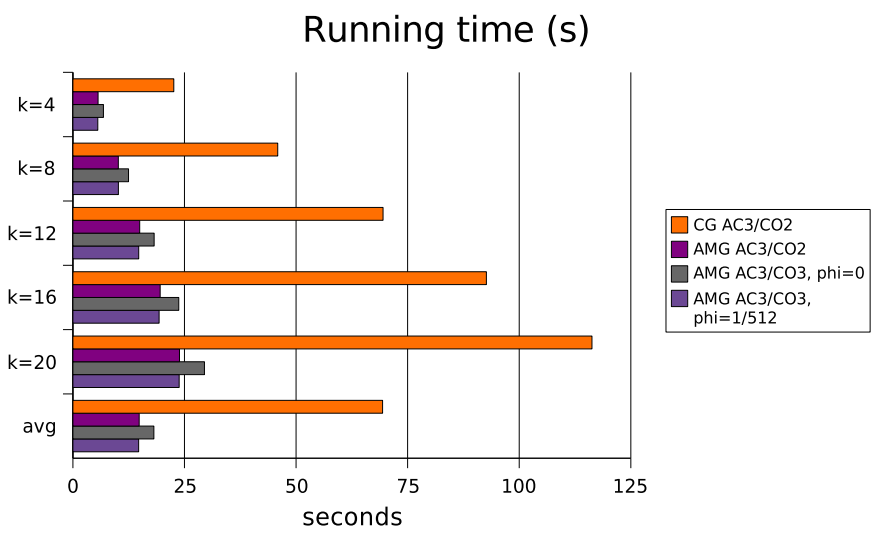


Figure 4.9.: Running time (in seconds) comparison between different linear solvers in Bubble-FOS/C (two upper bars in each row) and between FOS/C and FOS/V (two lower bars) as similarity measure within Bubble.

interest how the running time and the quality are affected if the virtual vertex scheme FOS/V is used instead of FOS/C. Therefore, we have repeated our experiments, but this time the virtual vertex and a higher CO iteration count is used. The FOS/V parameter ϕ is fixed to 1/512 because the main conclusions drawn from the results are very similar if ϕ is not varied too much from this setting. (Recall that ϕ should not be chosen too large or too small.) Note that AMG and CG are influenced very similarly w. r. t. running time and quality in our experiments, so that details of the latter are omitted.

As shown in Figures 4.9 and 4.10, the speedup by using the virtual vertex notion is consistently around 23% for $\phi = 1/512$. Interestingly, the quality of the results is also improved for the summation norm. This can be concluded from Figure 4.8 (and Table A.4 in the appendix), which shows the quality values in the two rightmost bars of each category for both AMG approaches, with FOS/C and with FOS/V. On the other hand, for the maximum norm a slight decrease in solution quality is also apparent.

k	Speedup AMG vs CG	Speedup FOS/V vs FOS/C
4	4.01	1.23
8	4.51	1.22
12	4.64	1.23
16	4.74	1.23
20	4.87	1.24
avg	4.67	1.23

Figure 4.10.: Running time speedups obtained by using AMG instead of CG (left) and FOS/V instead of FOS/C (right).

In summary AMG and FOS/V yield a running time improvement with a factor of about five to six compared to our previous work with Schamberger, while the decrease in solution quality is negligible.

Table 4.2.: Comparison of Bubble-FOS/C using with kMeTiS and Jostle for ℓ_1-norm and $k = 16$, detailed for each graph.

	kMeTiS		Jostle		Bubble-FOS/C	
Graph	EC	bnd	EC	bnd	EC	bnd
airfoil1	551.2	550.6	**541.2**	**537.2**	555.8	556.9
crack	1251.8	1231.1	**1182.4**	**1160.4**	1220.0	1197.6
whitacker_dual	640.2	1271.6	624.9	1239.2	**591.2**	**1149.2**
biplane9	822.8	1403.2	**779.5**	1408.8	813.3	**1241.0**
stufe10	**712.8**	1167.7	769.0	1289.0	748.0	**939.8**
altr4	7759.7	4551.5	7608.8	4457.2	**7214.1**	**4206.6**
shock9	1247.6	2099.4	**1129.4**	1980.0	1169.9	**1766.3**
wing	**4611.6**	8277.7	4639.4	8315.5	4765.8	**7521.4**

Table 4.3.: Comparison of Bubble-FOS/C using with kMeTiS and Jostle for ℓ_∞-norm and $k = 16$, detailed for each graph.

	kMeTiS		Jostle		Bubble-FOS/C	
Graph	ext	bnd	ext	bnd	ext	bnd
airfoil1	**97.4**	**48.9**	104.4	50.9	105.3	52.6
crack	221.6	108.7	211.7	103.5	**209.6**	**103.0**
whitacker_dual	110.1	108.9	108.5	107.5	**101.8**	**97.2**
biplane9	150.2	125.8	147.4	131.1	**144.6**	**103.1**
stufe10	130.6	106.9	172.2	141.7	**111.8**	**69.2**
altr4	1291.3	373.8	1260.8	362.9	**1110.2**	**318.7**
shock9	223.3	186.5	204.9	179.1	**200.1**	**143.1**
wing	790.9	697.4	902.7	781.5	**713.3**	**559.0**

4.7.2.4. Comparison to METIS and Jostle

Next, we evaluate our algorithm Bubble-FOS/C against METIS (more precisely kMeTiS 4.0[1] [Kary 98b], which implements direct k-way KL/FM improvement) and Jostle 3.0[2] [Wals 07a] because these two are state-of-the-art KL/FM partitioners. They are probably also the most popular general purpose sequential graph partitioners due to their speed and adequate quality. Both are used with default settings, so that their optimization objective is the edge-cut. We allow all programs to generate partitions with at most 3% imbalance, i. e., whose largest partition is at most 3% larger than the average partition size. To specify this is important because a higher imbalance can result in better partitions. Note that Scotch [Pell 07b] is not included in our presentation since our experiments show that kMeTiS delivers on average comparable or better results and is significantly faster. As can be expected for any recursive bipartitioning approach like Scotch, the loss in quality becomes particularly significant for larger k.

[1] The variant of METIS which yields shorter boundaries than kMeTiS is not chosen because its results show much higher edge-cut values than kMeTiS.

[2] Our experiments indicate that release 3.0 yields the same or comparable results as the latest release 3.1.

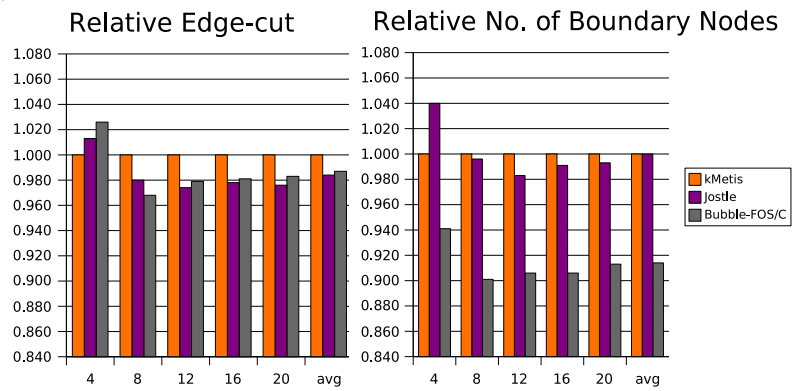


Figure 4.11.: Partitioning quality of JOSTLE and BUBBLE-FOS/C relative to KMETIS in the ℓ_1-norm for different k.

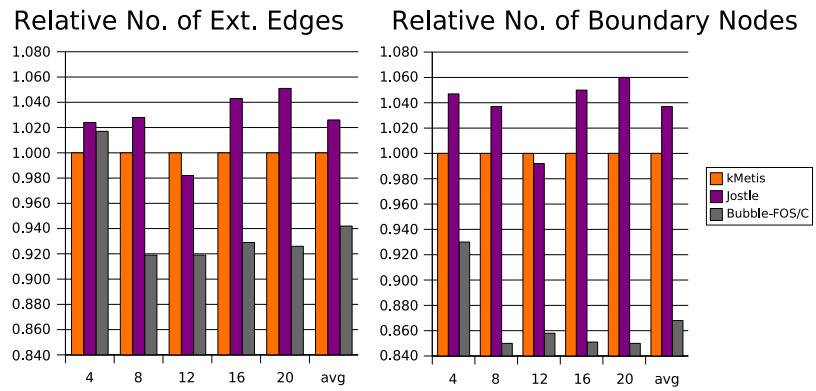


Figure 4.12.: Partitioning quality of JOSTLE and BUBBLE-FOS/C relative to KMETIS in the ℓ_∞-norm for different k.

The first comparison between BUBBLE-FOS/C – here represented by the parameter setting AC3/CO3 with virtual vertex and $\phi = 1/512$ – and its KL/FM counterparts shows the detailed average values for each graph obtained in ten runs on the benchmark set for $k = 16$. Table 4.2 displays the results in the summation norm, while Table 4.3 shows them in the maximum norm. The summation norm results reveal that BUBBLE-FOS/C is able to compute partitions with the best total number of boundary nodes in most cases. There is no clear winner w. r. t. the edge-cut, but JOSTLE obtains most best values (four out of eight). In the maximum norm BUBBLE-FOS/C is clearly the best. Except for the smallest graph airfoil1, it attains the best results regarding both the number of external edges and boundary nodes.

In order to estimate how the quality of the three programs relates to each other over a variety of values for k, we adopt the following evaluation scheme. For all values obtained for a graph (time, external edges, and boundary nodes) we use the results of KMETIS as standard of reference. This means that each value of the other two partitioners is divided by the respective value of KMETIS. Then, for each k and each metric, an average value of these ratios over all graphs is computed. These average values are displayed in

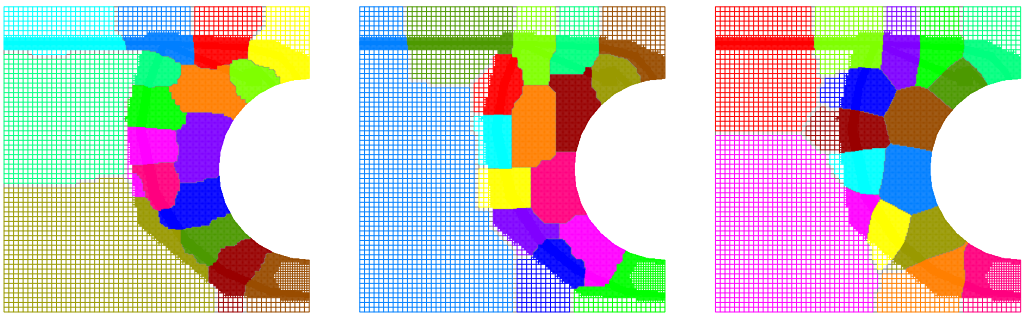


Figure 4.13.: Partitioning the *shock9* graph into 16 subdomains. The solution of METIS (left) shows jagged boundaries and elongated subdomain parts. JOSTLE's partition (middle) has somewhat smoother boundaries. BUBBLE-FOS/C (right) computes a solution where the shape-optimizing approach becomes apparent in the nearly convex subdomains.

Figures 4.11 (summation norm) and 4.12 (maximum norm) in the rows for the respective k. The column termed *avg* contains the respective average value of the averaged ratios. In this way each graph enters the averaging process equally to ensure a fair comparison.

Clearly, our algorithm BUBBLE-FOS/C is able to compute the partitions with the shortest boundaries, both in the summation and the maximum norm. It is also able to compute partitions with the fewest maximum external edges except for $k = 4$. The traditional edge-cut metric is best optimized in most cases by JOSTLE, but BUBBLE-FOS/C is not far behind. It is therefore possible to conclude from these data that the partitions computed by BUBBLE-FOS/C show the best overall properties compared to its competitors, at least for $k \in \{8, 12, 16, 20\}$. The largest improvement can be seen for the maximum number of boundary nodes, the metric which probably measures best the communication costs of parallel numerical solvers. In this metric our algorithm is 13.2% better than KMETIS and 16.3% better than JOSTLE.

Regarding the shape of the subdomains, Figure 4.13 makes some of the major differences between the three programs visible. In particular KMETIS computes solutions with jagged boundaries. JOSTLE on the other hand seems to aim at rectangular shapes, which is a good idea for edge-cut minimization. For short boundaries, however, convex subdomains are preferable. Although not all subdomains in the solution of BUBBLE-FOS/C are convex, the shape optimizing approach based on FOS/C is certainly recognizable. We are also interested in the diameter and connectedness of the subdomains. BUBBLE-FOS/C computes partitions with disconnected subdomains only in 3.5% of our test runs. In contrast to this, KMETIS and JOSTLE perform considerably worse, as they produce disconnected subdomains in 11.3% and 14.5% of the cases, respectively. To compare the diameter of the partitions, we evaluate the experiments for a medium number of subdomains, $k = 12$. Both in the summation norm and the maximum norm, BUBBLE-FOS/C computes solutions whose diameter is the smallest on average. Compared to JOSTLE, which is on average better than KMETIS in this category, our algorithm performs 4.6% (ℓ_1) and 10.5% (ℓ_∞) better, respectively.

Table 4.4.: Comparison of BUBBLE-FOS/C in different parameter settings for ℓ_1- and ℓ_∞-norm and $k = 2$.

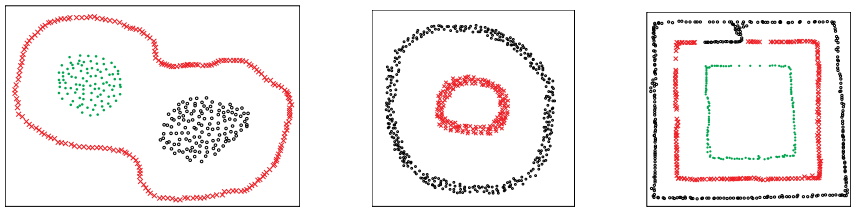
AC2/CO2			AC3/CO2			AC3/CO3		
EC	bnd_1	bnd_∞	EC	bnd_1	bnd_∞	EC	bnd_1	bnd_∞
2495.54	1630.39	904.33	**2390.70**	**1573.83**	**849.96**	2525.43	1652.86	910.11

The running times of the three programs are clearly in favor of KMETIS and JOSTLE. KMETIS requires only approximately two hundredth of a second to partition the graphs of the benchmark set. The other KL/FM partitioner JOSTLE is about 2.5 times slower than KMETIS, which is still very fast. Compared to these state-of-the-art libraries, BUBBLE-FOS/C requires much more running time. The values range from 5.58s for $k = 4$ to 23.81s for $k = 20$. Although we have sped up the algorithm by a factor of more than five compared to its previous state without AMG and the virtual vertex, it is still up to three orders of magnitude slower than its established competitors.

4.7.2.5. Influence of k

Our experiments indicate that BUBBLE-FOS/C in its current form is not very suitable for bipartitioning, i. e., when $k = 2$. A comparison of the data displayed in Table 4.4 with the values in Tables A.1 and A.2 reveals surprising results: For $k = 2$ the average edge-cut values in the summation norm ℓ_1 are much higher than for $k = 4$ and even higher k. Since the number of external edges typically increases with increasing k, this observation leaves only the conclusion that BUBBLE-FOS/C does not work properly for $k = 2$. One conjecture why this is the case is that the FOS/C load distributions interfere with each other in an unfavorable manner. Apparently, there are large regions with nodes whose load values for the two different subdomains are quite close together. Insofar, the decision to which subdomain they are assigned, is close to arbitrary. This effect of ambiguous affiliations is much smaller for larger k because such indecisive regions are smaller. Another reason might be a bad placement of the initial center nodes. While for large k it is likely to find a suitable initial spot for some centers, this might not be the case for bipartitioning. The iterative BUBBLE learning process is then not able to recover from such a situation.

Also, the number of subdomains k influences the running time of BUBBLE-FOS/C. Inspecting Figure 4.9, we see that this influence is close to linear because doubling k also (nearly) doubles the running time of BUBBLE-FOS/C. The reason for this is simple. If k is doubled, the number of linear systems to be solved is also doubled. Since this part of the algorithm is by far the most expensive one, the total running time is also nearly doubled. Such an effect cannot be observed with our two competitors. METIS and JOSTLE require hardly more time for partitioning if k is increased. That is why the running time gap between our algorithm and the other two libraries becomes larger and larger with increasing k. A remedy for this problem is of high interest, see Section 4.8.1.



(a) (b) (c)

Figure 4.14.: Three artificial data sets with intertwined clusters and their clusterings with the BUBBLE-FOS/C algorithm.

4.7.3. Graph Clustering

As the generic BUBBLE-FOS/C algorithm is meant for clustering into subdomains of arbitrary size, we would also like to know how it performs on clustering problems. For highly unstructured clustering problems such as random graphs the AMG hierarchy construction shows some deficiencies. The operator complexity, i. e., the total number of edges in the complete hierarchy, becomes quite high, which results in high running times. More importantly, the clustering quality is not satisfactory. That is why we have used the CG solver with the matching hierarchy instead. To aim at high-quality solutions, the operations `ComputeCenters` and `AssignPartition` are called alternately four times on each hierarchy level. All instances used in this section are rather small and should be seen as a proof-of-concept. For more clustering results refer to Section 5.4.3.

Recall that geometric k-means separates clusters only by hyperplanes. The reason is that the cluster affiliation is determined by the closest cluster center, so that essentially a Voronoi partition [Berg 97, Ch. 7] of the input is generated. When FOS/C is used instead of Euclidean distances, this limitation is no longer valid. This is shown in Figure 4.14. The cluster affiliation of the points in these three artificial 2D datasets are given as usual by their colors. Due to the intertwined structure of the clusters, k-means would fail for these instances, while BUBBLE-FOS/C works very well (except for a few debatable affiliations in the rightmost example).

Note that geometric datasets have to be transformed into edge-weighted graphs first, so that BUBBLE-FOS/C can work with them adequately. This transformation is begun by computing a minimum spanning tree between the input points to ensure connectedness of the graph. Additionally, for each node v, edges to the three nodes closest in Euclidean space to v are inserted. The weight of an edge $e = \{u, v\}$ is chosen proportional to e^{-x}, where x denotes the distance of u and v in Euclidean space. Hence, long edges get an extremely low weight. Note that we are aware of other algorithms, which can solve such simple planar instances equally well (e. g., geometric ones that build a minimum spanning tree and delete long edges [Page 74]). We have included these simple 2D examples nevertheless to show the differences one experiences, when the same algorithmic framework, but different distance/similarity measures are used.

The second experiment consists in a simple community detection problem. It uses

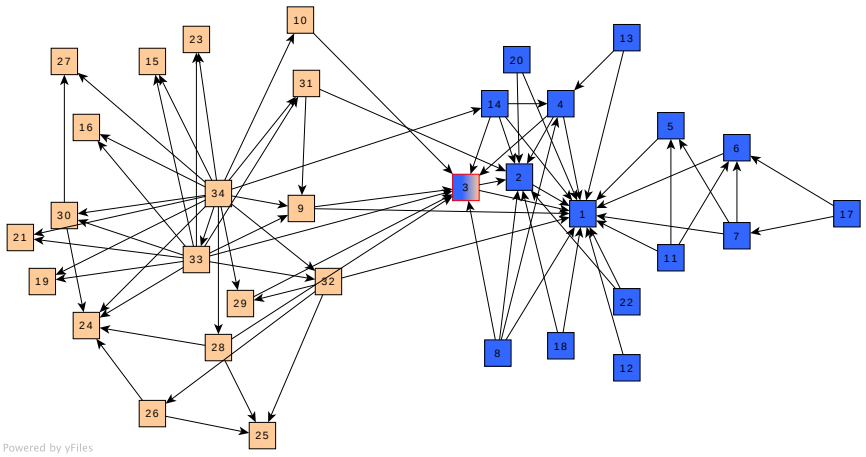


Figure 4.15.: Clustering of Zachary's karate club graph. (Note that, although the edges are drawn in a directed manner, the graph is in fact undirected.)

Table 4.5.: Normalized cut values of BUBBLE-FOS/C and GRACLUS on randomly generated graphs.

	NCut (GRACLUS)			NCut (BUBBLE-FOS/C)		
n	k=6	k=8	k=12	k=6	k=8	k=12
2^{11}	**0.328**	**0.641**	**0.704**	0.367	1.042	0.771
2^{12}	**0.323**	**0.626**	**0.678**	0.370	1.404	0.868
2^{13}	**0.331**	**0.639**	**0.796**	0.511	1.674	0.882
2^{14}	**0.335**	**0.641**	**0.716**	0.477	1.296	1.218

the real-world example of Zachary's karate club [Zach 77] with 34 nodes that form 2 clusters. The correct clustering is known and shown in Figure 4.15 by different colors (light orange and blue). Our algorithm computes a clustering that nearly matches the correct one. Only the node labeled with 3 and shown in both colors (and framed in red) should belong to the blue cluster, but is put into the other one. Note that this is not necessarily a flaw in the heuristic because node 3 has eight edges in total, four are incident to the first and four to the second cluster.

Another group of experiments is performed on randomly generated undirected graphs following the idea of *planted partitions* [Jerr 98]. Such clustered graphs are generated by specifying the cluster sizes and probabilities for intra- and inter-cluster edges a priori. Then, the edges of the graph are determined by these probabilities.

In this set of experiments we have switched off the SMOOTH operation. On these highly irregular instances it worsens the solution instead of improving it. We compare the results of BUBBLE-FOS/C to those of GRACLUS. Recall that the latter has been implemented by Dhillon et al. [Dhil 07] and is based on their kernel k-means algorithm enriched with additional features such as local search.

The twelve clustering problems used here have four different graph sizes $n \in \{2^{10}, 2^{11}, 2^{12}, 2^{13}\}$ and three different cluster numbers $k \in \{6, 8, 12\}$. For our experi-

ments we determine the cluster sizes randomly such that the largest one is at most twice as large as the smallest one. Then, for each node we draw the number of intra- and inter-cluster edges from two different normal distributions. Finally, the corresponding edges are added uniformly at random. The parameters of the normal distributions for creating the planted partitions are as follows: $\mu_{int} = 4.3, \sigma_{int} = 1.1, \mu_{ext} = 0.3, \sigma_{ext} = 0.3$ for $k = 6$ and $k = 12$ and $\mu_{int} = 5.1, \sigma_{int} = 1.3, \mu_{ext} = 0.45, \sigma_{ext} = 0.35$ for $k = 8$, where μ_{int} is the mean intra-cluster degree, μ_{ext} the mean inter-cluster degree, and σ_{int} and σ_{ext} their respective standard deviations. This set of parameters results in node degrees between 1 and 12.

The results of these instances are shown in Table 4.5 and compare the normalized cut values of Bubble-FOS/C to those of Graclus (KKM). Graclus is consistently better than Bubble-FOS/C. Our algorithm has particular problems with the instances that have eight clusters, probably due to the higher inter-cluster degree. Moreover, the running time of Bubble-FOS/C is about two orders of magnitude slower than that of Graclus. The latter requires less than $0.1s$ on these small instances. To summarize, our graph clustering experiments with Bubble-FOS/C and Graclus show:

- It may happen, in particular when k becomes larger, that (at least) one cluster center is not placed correctly in the beginning. Hence, one planted cluster contains more than one center, while another planted cluster contains no center. The iterative learning process of Bubble-FOS/C seems to have difficulties to recover from such a misplacement, leading to solutions with an inferior normalized cut.
- A careful inspection of the clusterings reveals that if the normalized cut is not extremely higher than that of Graclus, the result computed by Bubble-FOS/C is a reasonable clustering that deviates not very much from the correct one.
- The comparison to Graclus shows that Bubble-FOS/C is significantly slower (similar to the speed gap experienced during the partitioning experiments compared to METIS and Jostle) and also worse in terms of quality.

More experiments on planted partitions with comparisons of Bubble-FOS/C to Graclus and our other algorithm DibaP can be found in Section 5.4.3. There, we also motivate the use of Graclus and give more details on the algorithmic settings used in the experiments.

It is clear that the speed of Bubble-FOS/C needs to be improved to be competitive – just like for graph partitioning. But unlike before, for graph clustering it is necessary to improve the quality, too. In particular a mechanism to escape from bad initial center placements is necessary.

4.7.4. Parallelism

Recall that our experimental data including the running time have been assembled on a dual-core machine using POSIX threads in our implementation of Bubble-FOS/C.

One might argue that a comparison to the sequential libraries METIS, JOSTLE, and GRACLUS is biased since they do not make use of threads. We would like to make a case against this argumentation. First, all programs are run on the same hardware. Most modern standard desktop processors come with at least two cores nowadays and the number of cores will increase over the years due to technological progress. Hence, it is highly advisable to exploit all of these computational capabilities in order to fully utilize this progress. To our knowledge, however, no thread-parallel versions of METIS, JOSTLE, and GRACLUS exist. A reason for this could be the inherent sequential parts within the KL/FM heuristic. Parallelism by threads would probably deliver only small running time improvements. Moreover, automatic parallelism offered by the compiler is enabled for all programs except GRACLUS, so that the KL/FM partitioners profit somewhat from the parallel capabilities of the hardware.

Our experiments with the benchmark graphs of Table 4.1 show that BUBBLE-FOS/C attains a speedup of about 1.3 on the employed dual-core processor. This means that the threaded implementation is a factor of 1.3 faster than the non-threaded version. On our dual-core machine, this speedup corresponds to an efficiency of 0.65. Although we have gained a 30% improvement over the serial version, such low values are hardly satisfactory. Based on our data, we believe that this behavior stems from thread and cache conflicts. A thread is not likely to finish the complete solution phase of its assigned linear system. Instead, the CPU scheduler replaces it by another thread to ensure fairness. Next time the thread is restarted, all its data need to be loaded from memory into cache. This is a relatively expensive operation, which may prevent a better speedup.

4.8. Load Balancing and Partial Graph Coarsening

It has been described that AMG is in principle an optimal linear solver regarding its computational complexity and that we have achieved a significant speedup compared to the related implementation of BUBBLE-FOS/C, which is based on a CG solver and a matching hierarchy. Nevertheless, our algorithm is still very time-consuming compared to established partitioning libraries. This is already true for medium-sized graphs with some tens of thousands of nodes. It even worsens for larger graphs. In a distributed-memory implementation the parallel speedup for graphs of medium size would be limited due to the unfavorable computation-communication ratio. An additional drawback for large graphs is the quite expensive AMG hierarchy construction. Moreover, the construction algorithm is not easy to implement efficiently for distributed memory parallelism. Usage of the CG version would not improve the speed because the convergence rate of the CG solver typically slows down with the system size. Hence, a straightforward parallelization of BUBBLE-FOS/C for a large number of processors would hardly be able to achieve totally satisfactory running times. Consequently, to increase the practical relevance of our algorithm as a repartitioner for balancing the load in adaptive numerical simulations, additional techniques for its acceleration are required.

The major reason for the high running time compared to the state-of-the-art is the global approach of Bubble-FOS/C. Although its multilevel scheme provides a reasonably good initial partition on each level, the improvement process is not localized. Instead, the k linear systems of each FOS/C procedure are solved on the whole graph. Observe, however, that nodes in regions far away from the source set are very unlikely to become its new center or to belong to the corresponding subdomain. Thus, for these regions it should be sufficient to work with an approximation of them with fewer nodes and edges to reduce the complexity.

4.8.1. Partial Graph Coarsening

The locality observation is exploited in the following to speed up the computations under certain circumstances. We will describe the method developed with Schamberger [Meye 06c] only briefly here. As will be shown, it is only successful in certain cases. Yet, we do not want to forgo its description since its idea of localization is very valuable. It is also helpful to understand the alternative coarsening method we propose afterwards.

Assume that k FOS/C procedures have to be solved on a large input graph. First, a multilevel hierarchy based on approximate maximum weighted matchings is built. The procedures are then projected onto the coarsest level of this hierarchy, where they are solved by a standard solver such as CG. Figure 4.16 (left) illustrates a solution for one linear system on the coarsest level. Its highest load values (red color) can be found around the originating source node. Since the coarsening process has preserved the general graph structure, the load distributions on the lowest level can be expected to have a similar shape as the load distributions on the original graph. Hence, we are able to use them to determine the most relevant parts of the solution. To do this, the solutions are interpolated back onto the original graph. There, the nodes are classified based on their interpolated FOS/C loads into the three categories *high*, *medium*, and *low*.

The next step is performed independently for each linear system. Based on the categories, a new graph is assembled with different hierarchy levels of the original graph. Regions with high loads are carried over uncoarsened. In contrast to this, for regions with low values their approximation of the coarsest hierarchy level is used. Regions with medium loads are represented by a medium level of the hierarchy. Since this results in a graph with some coarsened and some uncoarsened areas, the method is termed *partial graph coarsening* (PGC). The original linear system is then projected onto the partially coarsened graph and solved. An example of a partially coarsened graph is given in Figure 4.16 (middle). The colors represent an FOS/C load distribution that has been calculated with such a varying accuracy.

The solution of the system with varying accuracy is then evaluated for `AssignPartition` or `ComputeCenters` in the usual manner. Depending on how strongly the graph has been coarsened, this modified solution process is reduced in complexity

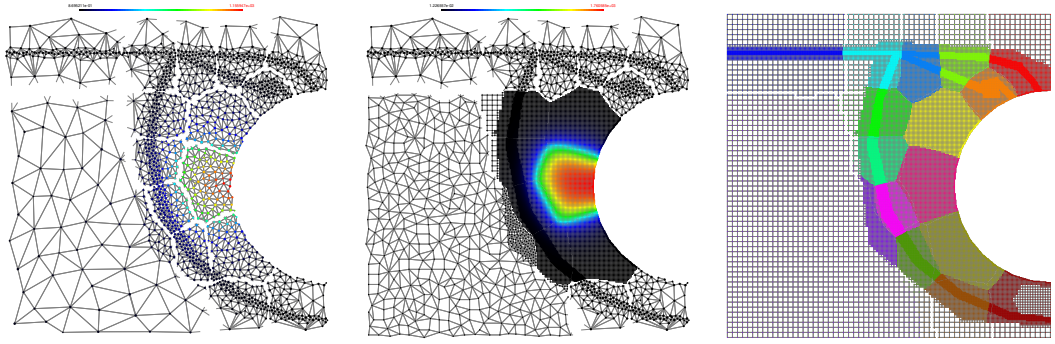


Figure 4.16.: Illustration of partial graph coarsening: Left: Vertex loads (colors from high to low: red, orange, yellow, green, cyan, blue, black) of one linear system on the lowest level. Middle: Final solution of the linear system on a graph assembled with varying accuracy. This solution has been computed for the pink subdomain in the displayed partition (right). Cut edges of the initial partition are not drawn completely.

compared to the generic one, although k additional (but small) linear systems have to be solved. Particularly for large k the additional work performed is supposed to pay off because the largest linear systems are reduced in size. By this means, the approximately linear scaling in k of the running time can be expected to be removed or at least eased. Note that for each of the k linear systems a different part of the graph is important. Hence, for each part different hierarchy levels contribute to the respective solutions.

In a distributed parallel setting, the graph and also the linear systems with varying accuracy are already stored in a distributed fashion over the processors. Hence, it would be natural to employ a parallel solver acting on these data in a distributed fashion. With PGC, however, another approach is viable. Since each linear system of varying accuracy is smaller than the original input graph, it can be sent to one of the processors. More precisely, each processor sends its part of the linear system i to processor i ($0 \leq i < k$). Then, processor i solves linear system i with a sequential linear solver, which does not require any communication. Afterwards, the computed load values are sent back to the processors of their originating parts. This approach is termed *domain sharing*. Its main difference to the standard approach is that only two large communication operations take place, before the solution process and afterwards, instead of three communication necessary within *each iteration* of a standard parallel CG solver.

4.8.2. Load Balancing Experiments

To evaluate PGC, load balancing experiments have been conducted with an MPI parallel implementation of Bubble-FOS/C on a cluster system with 200 computing nodes, each of which has two Intel Xeon 3.2 GHz EM64T processors and 4 GB RAM. In our tests we have used only one processor per node and for parallel communication the Scali MPI implementation via the Infiniband interconnection. The test graphs are a number of different 2D and 3D FEM graphs of different sizes. The task to be performed by Bubble-

FOS/C is to repartition the initial partitions of these graphs. Note that the number k of subdomains matches the number of processing nodes used, $k \in \{8, 16, 32, 64\}$. The initial partitions are provided by the partitioning library PARTY [Moni 04].

Two different versions of our algorithm are compared to evaluate the partial graph coarsening approach. Both perform the same number of outer and inner loop iterations within the BUBBLE-FOS/C algorithm. The first version is the standard algorithm for graph repartitioning without partial graph coarsening and without any multilevel hierarchy. It uses a standard parallel CG solver with a distributed graph data structure. The new scheme employs the partial graph coarsening approach, where the coarsening is done with approximate maximum weighted matchings. It also uses the domain sharing communication scheme. Several important parts of the implementation are due to Schamberger [Meye 06c, Scha 06]. That is why we address only important outcomes regarding PGC, which can be summarized as follows.

- All repartitioning tasks for graphs with up to half a million nodes and four million edges can be performed within two minutes with the respective fastest algorithm version.

- Doubling k also results in doubling the number of processors in our experimental setting. Yet, since the communication overhead increases with larger k, the running time of the generic algorithm still becomes larger with increasing k.

- If $k \leq 16$, most subdomains are still close to the respective source sets. Thus, the partial graph coarsening process is not very effective, as most subdomains are not coarsened at all. The additional work invested is not compensated in these cases, so that for small k the generic approach is still faster than PGC.

- With a large number of subdomains ($k \geq 32$) it is likely that a large fraction of them can be coarsened quite well with PGC. Indeed, as anticipated, the experiments show a significant running time improvement for PGC with domain sharing in these settings. While the running time of the generic algorithm generally becomes slower if k is increased, the new algorithm version speeds up with increasing k. Thus, the new method is able to alleviate the problem of the approximately linear dependence of the running time on k. Moreover, it is up to 4 times faster than the generic approach. Both PGC and the domain sharing communication pattern contribute to this acceleration.

- Regarding the partitioning quality and migration volume, BUBBLE-FOS/C tends to achieve better results than the parallel version of METIS. Compared to parallel JOSTLE, similar quality values and migration costs can be observed. Clearly in favor of BUBBLE-FOS/C is the shape of the subdomains and their connectedness.

4.8.3. A Possible Enhancement by Adaptive Graph Coarsening

The experimental results sketched above show that partial graph coarsening combined with domain sharing is successful for larger values of k. One of the main problems of Bubble-FOS/C, its linear dependence on k, can be weakened in this way. It is, however, for k around 16 and smaller slower than the generic approach without a multilevel hierarchy. Such settings do not allow a significant coarsening with PGC. More importantly, considering how many processors are involved in the computations, PGC with domain sharing is still quite time-consuming compared to state-of-the-art load balancers such as the parallel versions of METIS and Jostle.

The major reason for the moderate acceleration of the expensive generic Bubble-FOS/C by partial graph coarsening and domain sharing is that the localization of the approach is not always successful. Only if large regions can be identified as "far away" from the source set, the main part of the computational work of the solver is concentrated around the nodes of the source set. Yet, a more aggressive coarsening scheme is viable. Observe that very accurate FOS/C loads are only needed in those areas that are highly relevant for the assignment of nodes to subdomains or for computing a new center node. For `AssignPartition` these areas are the boundaries of the subdomains to be computed, while for `ComputeCenters` it is the region around the new center. In all other areas, the load values are so different that a correct choice of subdomains/centers can be made without a high precision.

Our new proposal of an *adaptive graph coarsening* scheme works as follows: The first part consisting of uniform coarsening and solving the linear system projected onto the coarsest level is equivalent to the related approach PGC. After that, the determination of the relevance of parts is modified significantly. For each node one computes based on its k FOS/C load values how often it may be merged during the matching coarsening process. Nodes whose two (or more) highest load values are close together must not be merged and therefore remain uncoarsened. The larger the difference between the highest and the other load values of a node v is, the more often v may take part in the coarsening process. Then, the matching algorithm used before, but modified to take the maximum level number of nodes into account, is employed to compute an approximation of the linear system. The relevance and therefore the maximum number of valid coarsening steps for a node change only slowly with its distance from the most important parts. That is why this approximation by adaptive coarsening is characterized by smoother transitions in the graph resolution.

Experiments with a draft implementation of the latter scheme indicate that the main idea of coarsening irrelevant parts without modifying the final partitioning result works quite well. The resulting linear systems are significantly more coarsened than with the related PGC approach, but still usable for Bubble-FOS/C. However, this new coarsening scheme is more complicated than the one used before. A thorough implementation and integration into Bubble-FOS/C has therefore not been conducted yet.

4.9. Discussion

The integration of FOS/C (or FOS/V) into the BUBBLE framework yields an iterative graph clustering/partitioning algorithm, which has the nice theoretical property to always converge towards a local optimum. On vertex-transitive graphs the algorithm computes connected subdomains, and on the two-dimensional torus it even finds the globally best solution (neglecting discretization errors). Our implementation with FOS/V and algebraic multigrid, both as linear solver and as a means for multilevel hierarchy construction, is about five to six times faster than related BUBBLE-FOS/C implementations. Experimental comparisons against the cutting-edge partitioning libraries METIS and JOSTLE can be summarized briefly as follows: BUBBLE-FOS/C's solutions attain fewer boundary nodes and often also a smaller edge-cut, in particular in the maximum norm. Yet, due to the high running time (BUBBLE-FOS/C is two to three orders of magnitude slower than the state-of-the-art), its practical relevance for high-performance simulations is doubtful. For graph clustering problems not only the running time of BUBBLE-FOS/C is a problem. Also the solution quality needs to be enhanced to be competitive to the kernel k-means based program GRACLUS.

The running time problem is partially addressed by two coarsening schemes (partial and adaptive graph coarsening). They follow the idea to coarsen irrelevant areas of the graph to solve linear systems only on an approximation of them. For large k this results in computations that are mostly concentrated on limited local areas of the graph. Yet, despite a more complicated algorithm, a large gap to the state-of-the-art libraries regarding running time remains. Insofar it is of utmost interest to design a faster *and* simpler algorithm. It should retain the main ideas of BUBBLE-FOS/C by using diffusive arguments for partitioning and of adaptive graph coarsening by a local improvement approach which concentrates on the respective most relevant graph regions. Such an algorithm is presented in the next chapter.

5. Faster Diffusion-based Partitioning

We have seen in the previous chapter that our algorithm Bubble-FOS/C, which combines disturbed diffusion with the Bubble framework, computes high-quality graph partitions with good shapes (whereas its clustering quality needs to be improved). Its disturbed diffusion scheme FOS/C is, however, quite expensive to compute compared to established partitioning heuristics – even after the introduction of our acceleration techniques. Consequently, the running time of our partitioning algorithm is too slow for real practical value. Although the global approach of solving many linear systems on the whole graph has been identified as Bubble-FOS/C's major drawback, the proposed coarsening approaches for a stronger localization have been only partially successful. While multigrid coarsening becomes very slow for large graphs, partial graph coarsening is only effective for a large number of subdomains, and adaptive graph coarsening requires a very complex and challenging implementation, particularly in parallel.

To overcome these limitations, we design in this chapter a fast and truly local algorithm to improve partitions generated in a multilevel process. It is simple to implement and exploits the observation that, once a reasonably good solution has been found, alterations during a local improvement step take place mostly at the subdomain boundaries. Like Bubble-FOS/C, the new algorithm TruncCons is also based on disturbed diffusion, where the disturbance is realized by truncating the diffusion process after a small number of iterations. This truncation allows for a concentration of the computations around the subdomain boundaries, where the changes in subdomain affiliation occur. Since also no linear systems need to be solved, no algebraic multigrid hierarchy has to be constructed. Instead, a much faster matching algorithm can be used to create a multilevel hierarchy for a successive improvement of the solution.

The initial solution on the coarsest hierarchy level of our new method is provided by Bubble-FOS/C. In fact, we can choose among several Bubble-FOS/C solutions to continue only with the best one. We will see later on that such a selection process alleviates problems that come from bad initial center placements. The combination of Bubble-FOS/C and TruncCons is called DibaP and, with some problem-specific extensions, performs very well in practice, as we show by extensive experiments.

5.1. A New Local Improvement Method: TruncCons

Recall from the previous chapter that the `Consolidation` operation is used to determine a new partition Π from a given one (cf. Figure 4.4 in Section 4.5). That is why it can be

seen as a mixture of `AssignPartition` and `ComputeCenters`. As shown in Algorithm 5 (lines 2-9), one `Consolidation` operation performs the following independently for each partition π_c: First, the source set S is initialized with π_c and the nodes of π_c receive an equal amount of high initial load, e. g., $n/|S|$. In contrast to that, the other nodes' initial load is set to a low value, e. g., 0 (lines 3-5). Then, a disturbed diffusive method is used to distribute this load within the graph. In the previous chapter we have used FOS/C, but this should be avoided due to its high running time – unless the graph is small.

To restrict the computational effort to areas close to the partition boundaries, we use here a small number ψ of FOS iterations (cf. Section 2.2) for distributing the load (lines 6-7). Hence, the disturbance comes from stopping the diffusive scheme long before convergence is reached. Although it corresponds to Schamberger's discarded FOS/L [Scha 06, p. 73], we call this diffusive method *truncated first order diffusion scheme (FOS/T)*. It can be written more formally as:

Definition 5.1. (FOS/T) Given a graph $G = (V, E, \omega)$, a source set $\emptyset \neq S \subset V$, and suitably chosen constants $\alpha > 0$ and $\psi \in \mathbb{N}$. Let the initial load vector $w^{(0)}$be defined as

$$[w^{(0)}]_v = \begin{cases} \frac{n}{|S|} & v \in S\,, \\ 0 & \text{otherwise}\,. \end{cases}$$

Then, the final load of a node v is obtained by ψ FOS iterations. More precisely, $[w_c^{(\psi)}]_v = [\mathbf{M}^\psi \cdot w_c^{(0)}]_v$, where $\mathbf{M}$ is the (possibly edge-weighted) diffusion matrix of G. This load situation can be computed by iterative load exchanges, too:

$$[w^{(t)}]_v = [w^{(t-1)}]_v - \alpha \sum_{\{u,v\}\in E} \omega_e([w^{(t-1)}]_v - [w^{(t-1)}]_u) \text{ for } 1 \leq t \leq \psi\,.$$

After the load is distributed with FOS/T for all k subdomains, we assign each node v to the subdomain it has obtained the highest load from (lines 8-9). This completes one `Consolidation` operation, which can be repeated several times to facilitate sufficiently large movements of the subdomains (cf. the **for** loop in line 1 of Algorithm 5). We denote the number of repetitions by Λ and call the whole method TruncCons (*truncated diffusion consolidations*).

Note that the Bubble operations `AssignPartition` and `ComputeCenters` are very problematic in connection with FOS/T, as indicated by Schamberger's related work on shape-optimizing graph partitioning [Scha 04a, Scha 06].[1] He has used FOS/T as a structural similarity measure for these other two Bubble operations.[2] Such an approach does not work well. For `AssignPartition` and `ComputeCenters` the choice of the number ψ

[1] That is why we have chosen not to call our new method Bubble-FOS/T.

[2] Note that in his early work [Scha 04a] on shape optimization Schamberger calls the `AssignPartition` operation `Consolidation`. Later, he uses `Consolidation` as the name for an operation that is a mixture of `AssignPartition` and `ComputeCenters` [Scha 06]. We have decided to adopt the latter nomenclature.

Algorithm 5 TruncCons ($\mathbf{M}$, k, Π, Λ, ψ) $\rightarrow \Pi$

```
01  for τ = 1 to Λ
      /* Begin Consolidation */
02    parallel for each π_c do
        /* Initial load */
03      S_c = π_c; w_c = (0, ..., 0)^T
04      for each v ∈ S_c do
05        [w_c]_v = n/|S|
        /* ψ FOS iterations */
06      for t = 1 to ψ do
07        w_c = M · w_c
      /* assign nodes to subdomains */
08    parallel for each v ∈ V do
09      Π(v) = argmax_{c∈{1,...,k}} [w_c]_v
      /* End Consolidation */
10  return Π
```

of FOS iterations is crucial. It is difficult to determine and should neither be too small nor too large in order to obtain meaningful load distributions. Hence, as pointed out by Schamberger himself [Scha 06, p. 73], in such a setting it is extremely doubtful if FOS/T can provide a practically useful non-balanced load distribution based on node connectivity. For the improvement of reasonable partitions with `Consolidation`, however, we will see that FOS/T is an excellent choice, which provides meaningful load distributions derived from disturbed diffusion.

5.1.1. Connection to Random Walks

To understand why one can expect TruncCons to work well, consider the following analogy. Recall that the stochastic diffusion matrix $\mathbf{M}$ can be seen as the transition matrix of a random walk. Let $S := \pi_c$ for some $c \in \{1, \ldots, k\}$, so that the source set is one of the current subdomains. Hence, we can assume that for each node $v \in S$ we have one random walk starting on v. Then, the final load on node u is proportional to the sum of the probabilities for each of these random walks to reach $u \in V$ after ψ steps. Since random walks need relatively long to leave dense regions, each node should be assigned to the subdomain with the highest load. With this subdomain it is most densely connected w. r. t. to the random walk notion described above.

5.1.2. Notion of Active and Inactive Nodes

Observe that during a TruncCons operation only certain nodes really take part in the iterative FOS/T load exchange of Definition 5.1.

Definition 5.2. A node $v \in V$ is called *active* in iteration $t > 0$ of an FOS/T procedure

if it has a neighbor $u \in V$ with the property: $[w^{(t-1)}]_u \neq [w^{(t-1)}]_v$. Nodes that are not active are called *inactive*.

All load exchanges of an inactive node result in a flow of 0 on its incident edges. Hence, they do not change the load situation at all and can be ignored. How to detect (most) inactive nodes, can be seen easily:

Observation 5.3. *Let $V_{inact}^{(t)} \subseteq V$ be the set of inactive nodes in iteration $t > 0$ of the FOS/T procedure invoked for a subdomain π_c during a* TruncCons *operation. Then, a node $v \in V$ is member of $V_{inact}^{(t)}$ if for every boundary node $u \in \pi_c$, we have:* $\text{dist}(v, u) > t - x$, *where $x = 0$ for $v \in V \backslash \pi_c$ and $x = 1$ for $v \in \pi_c$.*

This observation might not give away all inactive nodes since there might be nodes that are closer to the boundary and have only neighbors with the same load by coincidence. However, one can expect these cases to be very rare and most likely the vast majority of inactive nodes are covered.

So, by keeping track of active and inactive nodes using the above observation, we are able to ignore nearly all load exchange computations that do not change the respective loads on the incident nodes. In this way, the diffusive process of partition improvement is restricted to local areas close to the subdomain boundaries. The complexity of FOS/T is therefore greatly reduced in practice, although the effectiveness of the reduction depends on several factors. These are the iteration number t, the number of subdomains k, and the size and the structure of the graph. We can avoid more computations if t and k are small and the graph is large and sparse than in the opposite case.

5.1.3. Discussion of TruncCons

TruncCons retains the basic ideas of Bubble-FOS/C. It uses a disturbed diffusion scheme with a random walk connection to assign to each node a load value for each subdomain. This similarity makes it easy to integrate into our work, both conceptually and implementation-wise. As TruncCons does not require the solution of linear systems, AMG is not required. Hence, for providing a multilevel hierarchy we can use approximate maximum weighted matchings instead. Using such matchings is advantageous because they are much faster to compute for large graphs than the hierarchy construction of AMG, whose computation of the coarse matrices involves matrix-matrix multiplications.

It should be added that for small ψ (in our experiments we mostly use a value of $\psi \leq 25$) and sufficiently large graphs it is almost certain that TruncCons requires fewer iterations (of such simplicity) than a linear solver like CG for Bubble-FOS/C. Compared to AMG and its cycles, the amount of operations per iteration is greatly reduced. Thus, savings in the running time can be expected also versus the faster linear solver. Compared to the partial graph coarsening (PGC) method of Section 4.8, our new algorithm always achieves a true localization of the computational effort around the subdomain boundaries. In contrast to this, PGC succeeds in this only for large k.

Algorithm 6 GenericDibaP $(G = (V, E), k, \Pi,$ `thrsh`, `level`, $\Lambda, \psi) \rightarrow \Pi$

```
01  if |V| < thrsh then
02    Π = MultilevelBubble-FOS/C (G, k, Π)
03  else
04    [G', Π'] = MatchingCoarsen(G, Π)
05    Π' = GenericDibaP (G', k, Π', thrsh, level + 1, Λ, ψ)
06    Π = Interpolate(Π')
07  Π = TruncCons (M(G), k, Π, Λ, ψ)
08  if (level == 0) then
09    Π = Smooth(Π)
10  return Π
```

Another advantage of FOS/T and TruncCons is its simplicity. The load exchanges via the edges are very basic operations. Apart from the CPU, very fast dedicated hardware such as general purpose graphics processors can be employed for these computations. An implementation based on this idea is part of future work. The simplicity makes TruncCons also more appealing than adaptive graph coarsening (Section 4.8). While the latter also restricts the computational effort to limited areas, it is much more complicated to implement.

Observe that local improvements by TruncCons are inherently parallel processes. As with Bubble-FOS/C, computing the final disturbed diffusion load can be performed independently for each subdomain. Moreover, the determination of the maximum load for affiliating the nodes to subdomains is independent for each node. Even within each FOS/T iteration the load updates are independent for each node (if additional memory is used). Hence, both shared-memory and distributed-memory parallelizations can be expected to deliver noticeable accelerations.

In summary, one can say that our new algorithm makes Schamberger's ideas [Scha 04a] robust, practicable, and fast. Moreover, although showing some differences, it can be viewed as a k-way extension of the diffusive part in Pellegrini's work on shape optimization [Pell 07a] mentioned in Section 1.3.4. Now that our truly local scheme for improving partitions based on disturbed diffusion is available, we need to specify how to use it in the context of our previous work. Of course, the outcome is supposed to be much faster than Bubble-FOS/C. At the same time, the new algorithm needs to retain or improve Bubble-FOS/C's solution quality. The solution to this problem is addressed in the subsequent sections.

5.2. The (Re)Partitioning and Clustering Algorithm DibaP

5.2.1. Combined Hierarchies, Combined Algorithms

For the integration of TruncCons with Bubble-FOS/C we follow a well-known practice in computer science. Two algorithms that solve the same problem at different speeds

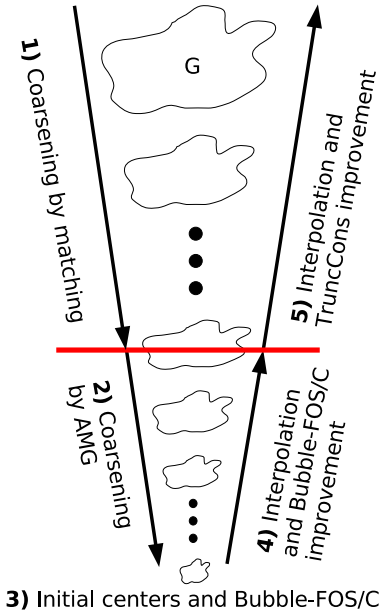


Figure 5.1.: Sketch of the combined multilevel hierarchy and the corresponding partitioning algorithms used within DibaP.

can be combined as follows. The slow algorithm (here Bubble-FOS/C), which has a favorable property (here it delivers a high solution quality – at least for (re)partitioning), is used for computing a solution to a coarse representation of the input. Afterwards, this coarse solution is fed into a faster algorithm (here TruncCons), which uses it as a starting point to solve the original problem. A similar combination concept described by Jájá in his textbook [JaJa 92, Ch. 2.6] is called *accelerated cascading*. In our case this combination yields an efficient multilevel graph partitioning algorithm that we call DibaP (*Diffusion-based Partitioning*).

As shown in Figure 5.1, the fine levels of DibaP's multilevel hierarchy are constructed by approximate maximum weight matchings (1). Once the graphs are sufficiently small, we switch the construction mechanism to the more expensive AMG coarsening (2). This is advantageous because, after computing initial centers and a partition with Bubble-FOS/C (3), we use the latter algorithm also as the improvement strategy on the next few hierarchy levels (4). Since AMG is employed to solve the occurring linear systems, such a hierarchy needs to be built anyway. On the finer parts of the hierarchy constructed by matchings, the faster TruncCons is used as the local improvement algorithm (5). A formal definition of DibaP is displayed as Algorithm 6. The initial call assigns 0 to the parameter `level`. Implementation details and additional operations such as balancing are discussed in Section 5.3.

Note that it is questionable whether TruncCons can be adapted to (re)partition or cluster graphs equally well without using Bubble-FOS/C before. Our experiments indicate that the subdomain shapes and other important properties of the solutions suffer in quality if TruncCons is used too early in the multilevel process or even exclusively.

The combination of the algorithms is therefore necessary to obtain speed *and* quality.

5.2.2. Computational Complexity

For sufficiently large graphs it is clear that the running time of DibaP is dominated by that of TruncCons. The reason is simply that the size of the hierarchy level on which the algorithm switch takes place can be fixed with a constant, the parameter `thrsh`. In that case the Bubble-FOS/C part of DibaP requires nearly always the same amount of time for the same amount of subdomains, regardless of the input graph size.

Within TruncCons one performs for each subdomain Λ times ψ FOS iterations. In the (unrealistic) worst case, for each edge of the graph a load exchange takes place in every iteration. Hence, in this case the running time is proportional to $k \cdot \Lambda \cdot \psi \cdot |E|$. The affiliation of nodes to subdomains requires $n \cdot k$ operations. If Λ and ψ are seen as constants, the asymptotic running time is bounded by $\mathcal{O}(k \cdot |E|)$. The linear dependence on the factor k – instead of an additive penalty for increasing the number of subdomains – can be seen as the major drawback in the running time of TruncCons. Furthermore, the product $\Lambda \cdot \psi$ might be quite large, depending on the user choice. On the other hand, due to the notion of (in)active nodes, the number of operations actually performed will be much smaller in practice. Since the savings depend on many factors that differ from input to input, a theoretical worst-case analysis is not likely to predict the final running time accurately. That is why we also refer to our experiments in Section 5.4.1.5, which essentially confirm our theoretical result of $\mathcal{O}(k \cdot |E|)$.

5.2.3. Multiple Coarse Solutions

The use of Bubble-FOS/C for computing an initial solution within the multilevel framework provides an interesting opportunity for avoiding bad initial partitions. Recall that in Bubble-FOS/C it is possible to select the most suitable set of initially chosen centers from a sample. A similar idea of choosing from multiple solutions can be pursued here. Before starting multilevel partitioning with TruncCons, we call Bubble-FOS/C a number of times and keep only the best of the solutions. Since the graph on the coarsest TruncCons level (the finest Bubble-FOS/C level) is relatively small, Bubble-FOS/C returns a solution quite fast. Experiments in Section 5.4.1 reveal several positive effects of this sampling approach. Which metric is used to determine the best solution, depends on the application and can be chosen by the user.

5.3. Problem-specific Adaptations and Implementation Details

As before, for graph partitioning and repartitioning the subdomain sizes have to meet specified balance constraints. Regarding this aspect, another advantage of DibaP (respectively TruncCons) becomes apparent. Like Bubble-FOS/C, it computes k load

values for each node based on a diffusive process. This similarity of the algorithms allows for an easy adaptation of the balancing methods described in the previous chapter for their use with TruncCons. The same applies to the operation Smooth, which is used on the finest level to straighten the subdomain boundaries of the final solution.

Whenever DibaP is used for repartitioning, one part of its input is an initial partition. We can assume that this partition is probably more unbalanced than advisable. It might also contain some undesirable artifacts. Nevertheless, its quality is not likely to be extremely bad, as it has emerged from a previous partition of good quality. It is therefore reasonable to improve the initial partition instead of starting from scratch. If DibaP is called, although the balance of the partition is below the imbalance threshold, we perform TruncCons with very small Λ and ψ (e. g., both are set to 3) on the input graph only. This is relatively inexpensive, but eliminates possible artifacts. It also improves the quality of the subdomains somewhat, while it generates hardly any migration costs. In the other case, which means that rebalancing is really necessary, the multilevel paradigm comes into play again. A matching hierarchy is constructed until only a few thousand nodes remain in the coarsest graph. The initial partition is projected downwards the hierarchy onto the coarsest level. This projection is done in the following manner. Recall that, due to the hierarchy construction by matchings, a node v in the graph of level $i+1$ has one or two parent nodes in the graph of level i. If all parent nodes are in the same subdomain π_c, v is also assigned to π_c. Otherwise, v is assigned to the subdomain of the parent node with the higher weight, where ties are broken arbitrarily. On the coarsest level the graph is partitioned with Bubble-FOS/C, starting with the projected initial solution. Going up the multilevel hierarchy recursively, the result is then improved with TruncCons and interpolated to the next level.

It may happen that the matching algorithm has hardly coarsened a level, in order to avoid star-like subgraphs with strongly varying node degrees. This limited coarsening yields two very similar adjacent levels. Local improvement with TruncCons on both of these levels would essentially result in the same work being done twice. That is why in such a case TruncCons is skipped on the higher level of the two (which is processed *after* the lower one in the improvement phase).

Keeping track of active nodes within TruncCons is currently done with an array, in which we store for each node its status. This could be possibly improved by a faster data structure that considers only the active nodes, such as a set based on hashing.

5.4. Experimental Results

There are three major parameters that control the quality and running time of DibaP. The first one is the switch threshold `thrsh`, denoting the size of the hierarchy level at which the switch between TruncCons and Bubble-FOS/C takes place. On the next-lower level, an initial solution is computed by multilevel Bubble-FOS/C and afterwards refined by TruncCons. In our experiments we have used different values for `thrsh`,

Table 5.1.: Graphs used in the experiments of Section 5.4.1.

	Size		Degree			
Graph	$\|V\|$	$\|E\|$	min	max	avg	Origin
tooth	78,136	452,591	3	39	11.585	FEM 3D
rotor	99,617	662,431	5	125	13.300	FEM 3D
598a	110,971	741,934	5	26	13.372	FEM 3D
ocean	143,437	409,593	1	6	5.711	FEM 3D
144	144,649	1,074,393	4	26	14.855	FEM 3D
wave	156,317	1,059,331	3	44	13.554	FEM 3D
m14b	214,765	1,679,018	4	40	15.636	FEM 3D
auto	448,695	3,314,611	4	37	14.774	FEM 3D

ranging from $1,000$ to $10,000$. The actual choices for the presented data depend on the application and are specified in the upcoming sections. It turns out that for graph clustering the `thrsh` value should often be smaller than for graph (re)partitioning. This probably results from the limited clustering solution qualities obtained with BUBBLE-FOS/C in random graphs.

The hard- and software environment for the graph partitioning experiments with DIBAP is the same as in Section 4.7. To exploit the parallelism offered by the dual-core processor, we have parallelized the computation of the disturbed diffusion loads within TRUNCCONS by POSIX threads. More precisely, for each subdomain one thread is responsible for computing the FOS/T load values. As discussed in Section 4.7.4, threads are not available for the programs that serve as standard of reference.

The test graphs in this chapter are much larger to better reflect typical input sizes for the utilized hardware. These graphs are a mixture of real-world instances, generated data that imitate real-world problems, and randomly generated graphs. Such a variety avoids the concentration on problems which are too similar to each other. Again, the graph partitioning and clustering experiments are repeated ten times for each graph. The repartitioning experiments are conducted only once per graph sequence. Since each sequence consists of at least 46 graphs, random influences are averaged over the whole sequence. As before, running times are given in seconds (unless stated otherwise) and best values within a table row are written in bold font.

5.4.1. Graph Partitioning

The graphs used in the graph partitioning experiments of this chapter are displayed in Table 5.1. They have been chosen because of their public availability from Chris Walshaw's well-known graph partitioning archive [Sope 04, Wals 07b]. Furthermore, they are the eight largest graphs therein w. r. t. the number of nodes and represent the general trends in our experiments. Hence, they constitute a good sample and also model large enough problems from three-dimensional numerical simulations (*598a* and *m14b* are meshes of submarines and *auto* of a car [Huan 06]).

Table 5.2.: Parameter settings and resulting running times in seconds of DibaP for partitioning the "average benchmark graph". Left: Influence of multiple coarse solutions. Right: Influence of loop parameters Λ and ψ.

Setting	Λ	ψ	#coarse	Time (s)
1a	10	14	1	16.52
2a	10	14	3	18.73
3a	14	19	1	36.51
4a	14	19	3	39.21

Setting	Λ	ψ	#coarse	Time (s)
1b	6	9	3	7.53
2b	10	9	3	10.53
3b	10	14	3	18.73
4b	14	19	3	39.21

In the experiments presented in this section, Bubble-FOS/C is used for all hierarchy levels with less than 5000 nodes. Larger levels are processed with TruncCons. Bubble-FOS/C performs two iterations of `ComputeCenters` and `AssignPartition`, followed by two `Consolidations`, and uses FOS/V as similarity measure ($\phi = 1/512$). The AMG multilevel coarsening is stopped when the graph has at most $48 \cdot k$ nodes.

5.4.1.1. Influence of Multiple Coarse Solutions

First of all, we would like to evaluate how strongly multiple coarse solutions computed by Bubble-FOS/C increase the average partitioning quality and if the running time is affected severely by this. That is why we have conducted a set of experiments where the loop parameters Λ and ψ are fixed, but the number of coarse solutions are varied.

Table 5.2 (left) displays the parameter settings and the resulting running times for partitioning the benchmark set with DibaP. The data are highly aggregated, as they are averaged over all graphs, different numbers of k ($k \in \{4, 8, 12, 16, 20, 32\}$), and all ten runs on each graph. In order to compare different parameter settings, such an aggregation is helpful since it reveals the trends within the data generated by the same basic algorithm. The quality obtained by DibaP in the four settings is shown in Figure 5.2. The first two columns show the aggregated values for the external edges and boundary nodes in the summation norm, respectively. Then, in the next two columns, the data for the maximum norm follow. Note that for presentation reasons the values in the first three columns have been divided by the factor shown in the x-axis.

The parameter settings $1a$ and $2a$ differ from each other in the number of initial coarse solutions computed by Bubble-FOS/C (one for the first one, three for the second one). This holds similarly for the settings $3a$ and $4a$. It is of course not surprising that the average solution quality is improved by choosing the best out of more than one initial solutions, as can be seen from the data. Similarly, that the most time-consuming parameter setting achieves the best quality meets our expectations. Yet, it is remarkable that the average quality of setting $2a$ is consistently better than of setting $3a$, although much less computational effort has been invested. This indicates that selecting a very good initial partition is also very helpful for saving running time. It is much cheaper than additional TruncCons operations, as can be seen by the large running time difference

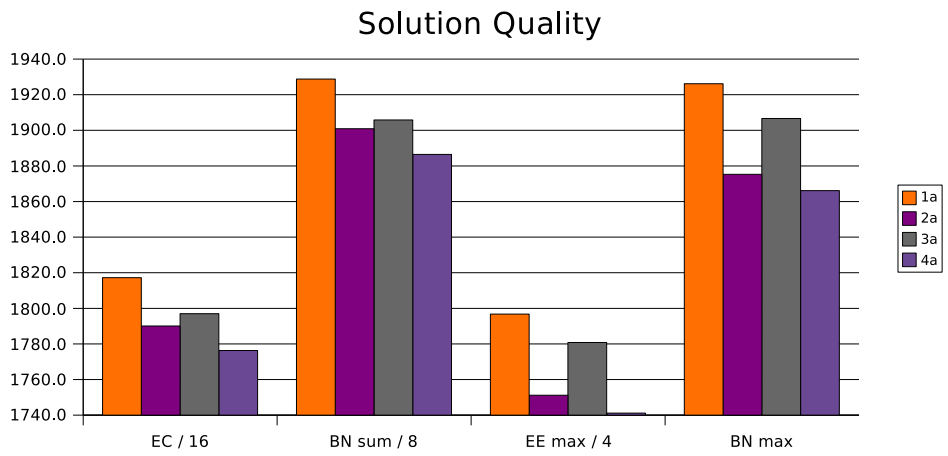


Figure 5.2.: Solution quality obtained by DIBAP in the four different parameter settings of Table 5.2 (left).

between settings $2a$ and $3a$. One can expect that the gain derived from multiple solutions declines with an increasing number of initial solutions. While three might not be the best choice, our experiments are only meant to show the general trend rather than an optimal value.

Another positive influence of having multiple choices is the reduced variance in the data. For example for $k = 12$, selecting the best of three initial solutions reduces the variance in the edge-cut by an average factor of more than six. Hence, the results are not only better on average, but also more reliable in the sense that they deviate less from the mean. That is why we use in the following experiments the best w. r. t. the edge-cut of three initial solutions generated by BUBBLE-FOS/C.

5.4.1.2. Different Loop Parameters Λ and ψ

Before we compare our new algorithm to other ones, we should find out which parameters Λ and ψ work well. For this we have selected four different combinations of them, as can be seen in Table 5.2 (right). Their solution quality is depicted in Figure 5.3, whose three first data columns are scaled for presentation reasons. Again, the most expensive version attains the best quality in all four metrics. Yet, setting $3b$ with $\Lambda = 10$, $\psi = 14$ is not too far behind, in particular not for the number of boundary nodes in the maximum norm (BN_∞). Considering that it requires less than half of the running time (cf. the right part of Table 5.2), it should be regarded as a good alternative in practice.

It becomes also apparent that a sufficiently large ψ is of really high importance. While the values on the partition quality of settings $1b$ and $2b$ do not deviate from each other very much, the increase of ψ from the second to the third setting results in a large quality improvement. (Of course, this also involves a large increase in running time.) If one is interested in a fast and only reasonably good solution, one can choose rather small values of Λ and ψ. To obtain a really high quality, however, $\psi = 9$ is apparently not enough.

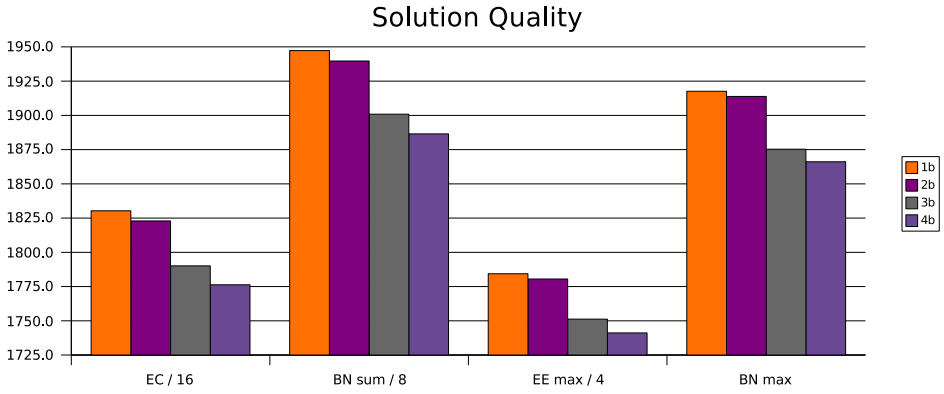


Figure 5.3.: Solution quality obtained by DibaP in the four different parameter settings of Table 5.2 (right).

Table 5.3.: Quality and running time of Bubble-FOS/C and DibaP for three different pairs of graphs and k.

		Bubble-FOS/C			DibaP		
Graph	k	EC	bnd_∞	Time (s)	EC	bnd_∞	Time (s)
tooth	8	12821.4	1287.0	43.29	**12394.4**	**1222.4**	**4.87**
144	16	41208.8	**1400.1**	222.67	**39431.0**	1484.1	**20.15**
m14b	32	75354.8	1347.5	820.08	**68611.2**	**1292.9**	**50.51**

For really excellent results the loop parameters have to be increased significantly, as it has been done in settings $3b$ and $4b$.

5.4.1.3. Comparison to Bubble-FOS/C

In order to show that DibaP constitutes a substantial acceleration, we compare its running time to that of Bubble-FOS/C. We also include some of the quality measures to estimate how the partitioning results differ. A complete run of Bubble-FOS/C on all large benchmark graphs would be extremely time-consuming. Since the general trend is always very similar, we have restricted our detailed experiments to a subset of the benchmark graphs and subdomain numbers k. Our presentation is restricted to a representative sample of this subset, see Table 5.3. It displays the edge-cut, the maximum number of boundary nodes, and the running time for Bubble-FOS/C and DibaP. For these experiments Bubble-FOS/C is called with parameters AC3/CO2 and $\phi = 1/512$, while DibaP uses the setting $\Lambda = 10$, $\psi = 14$.

The results show that DibaP is a factor of about 9 to 16 faster than Bubble-FOS/C. As could be expected, the speed gain increases with the size of the graph and k. Moreover, the edge-cut is also better with DibaP, while the maximum boundary length is sometimes better, sometimes worse than with Bubble-FOS/C. Since the values for the omitted two quality measures are in favor of DibaP, we conclude that in most cases it improves the partition quality compared to Bubble-FOS/C, and it is significantly faster.

Table 5.4.: Comparison of DibaP with kMetiS and Jostle for ℓ_1-norm and $k = 16$, detailed for each graph.

	kMetiS		Jostle		DibaP ($\Lambda = 10$, $\psi = 14$)	
Graph	EC	bnd	EC	bnd	EC	bnd
tooth	20408.1	11744.9	19619.0	11257.4	**18724.8**	**10745.6**
rotor	24118.6	12808.7	23898.4	12676.0	**22819.6**	**12063.7**
598a	29400.6	15255.1	28679.7	14841.1	**27057.4**	**13859.0**
ocean	10159.9	14876.4	**9106.7**	14929.7	9530.7	**13387.0**
144	42857.5	20533.5	41795.1	20016.8	**39431.0**	**18736.3**
wave	48101.1	24899.8	48504.9	25076.7	**44788.4**	**23130.7**
m14b	49207.8	22749.4	48234.5	22197.5	**44708.7**	**20409.8**
auto	87855.9	44293.3	90075.6	45234.3	**80298.8**	**40282.3**

Table 5.5.: Comparison of DibaP with kMetiS and Jostle for ℓ_∞-norm and $k = 16$, detailed for each graph.

	kMetiS		Jostle		DibaP ($\Lambda = 10$, $\psi = 14$)	
Graph	ext	bnd	ext	bnd	ext	bnd
tooth	3771.2	1069.7	4191.6	1171.1	**3256.2**	**904.6**
rotor	4255.4	1134.7	4451.6	1158.8	**4189.2**	**1100.2**
598a	5438.7	1391.5	5529.0	1398.0	**4696.6**	**1195.2**
ocean	1874.5	1359.4	1808.1	1472.8	**1758.2**	**1213.4**
144	7074.2	1679.2	7327.3	1740.1	**6227.0**	**1484.1**
wave	7800.4	2022.3	7980.0	2056.7	**7217.1**	**1862.5**
m14b	8853.5	2020.8	9077.8	2050.5	**7553.1**	**1723.9**
auto	15541.8	3917.8	16631.9	4099.2	**14686.7**	**3673.8**

5.4.1.4. Comparison to METIS and Jostle

Analogous to the previous chapter, we compare our new partitioning algorithm DibaP ($\Lambda = 10$, $\psi = 14$, unless stated otherwise) to the state-of-the-art libraries METIS and Jostle. (Recall that Pellegrini's Scotch is not included because our experiments have shown that Scotch is slower than kMetiS and does not deliver better solution qualities compared to kMetiS.) The first set of results is displayed in Tables 5.4 and 5.5, for which we have chosen $k = 16$. The values of the quality measures obtained on this representative sample are shown for each graph in detail; in the summation norm in Table 5.4, in the maximum norm in Table 5.5. Except for one value, the edge-cut of the graph *ocean*, DibaP is able to achieve the best values in all metrics and for all graphs. This is insofar remarkable as DibaP even obtains almost always better edge-cuts, although the underlying diffusive algorithms do not explicitly optimize this metric like kMetiS and Jostle using KL/FM.

Similar as before, we also present the aggregated quality values for different k. Figures 5.4 (summation norm) and 5.5 (maximum norm) show the average quality of partitioning the benchmark set *relative* to the values obtained by kMetiS. In both figures,

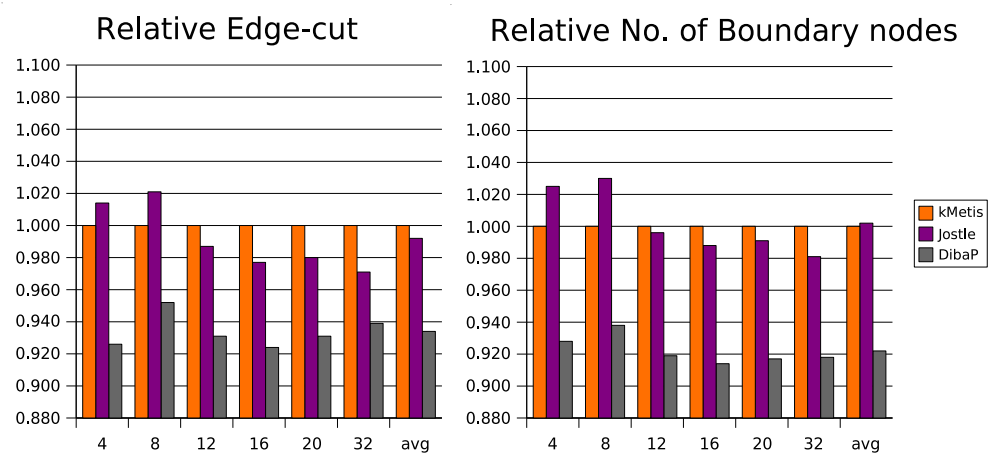


Figure 5.4.: Partitioning quality (ℓ_1-norm) of JOSTLE and DIBAP relative to KMETIS for different k.

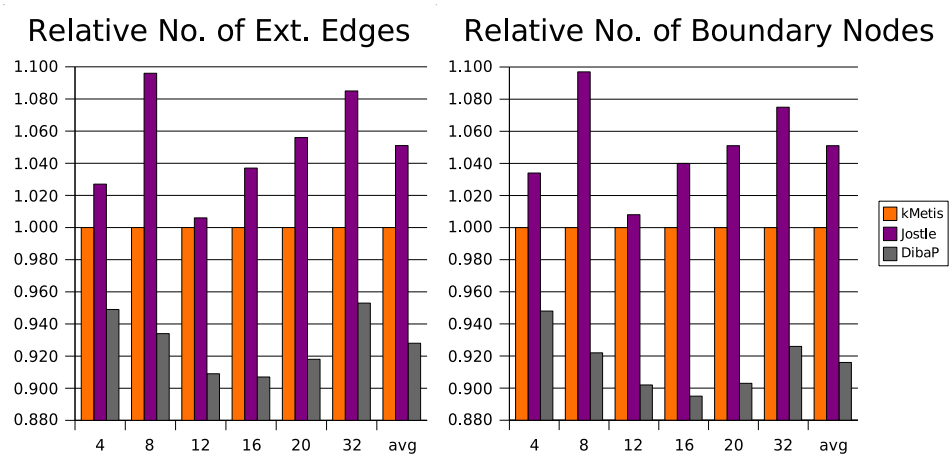
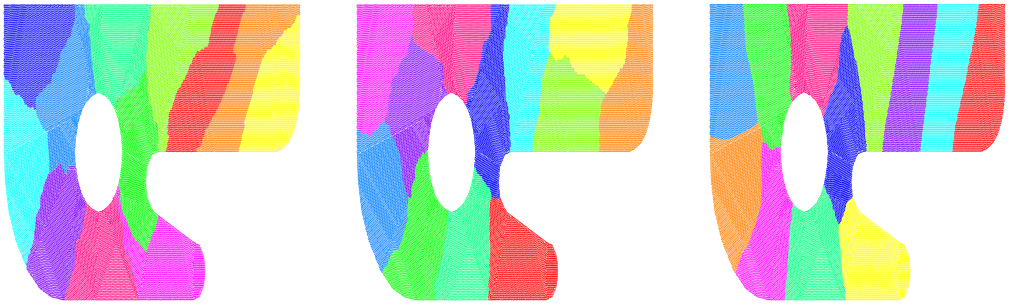


Figure 5.5.: Partitioning quality (ℓ_∞-norm) of JOSTLE and DIBAP relative to KMETIS for different k.

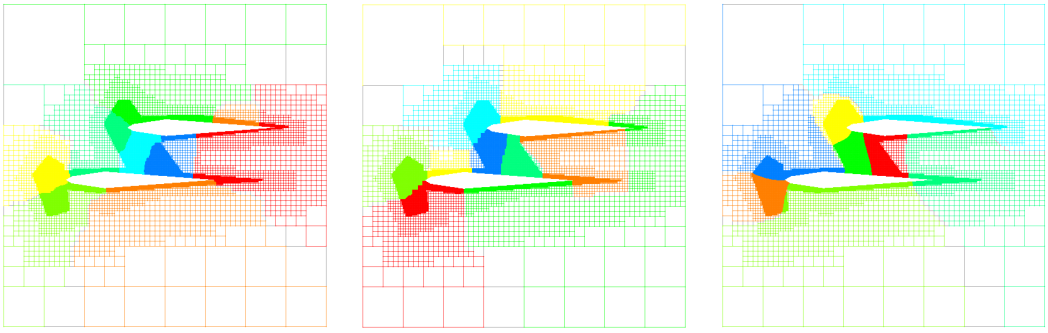
i. e., for both norms, our algorithm DIBAP obtains all possible best values for the different values of k. On average it beats KMETIS by 6.6% (edge-cut), 7.8% (sum of boundary nodes), 7.2% (maximum external edges), and 8.4% (maximum boundary nodes), respectively. Versus JOSTLE, these relative values are even a little bit better. Recall that if DIBAP is allowed to perform more iterations, e. g., $\Lambda = 14$ and $\psi = 19$, its average solution quality can be even further improved (cf. Figure 5.2).

To provide the reader with a visual impression on how DIBAP's results differ from those of KMETIS and JOSTLE, we include a 12-partitioning of the 2D graph *t60k* (also available from Walshaw's archive), see Figure 5.6. The partitioning computed by DIBAP ($\Lambda = 12$, $\psi = 18$) has not only fewer cut edges and boundary nodes in both norms than the other libraries. Its partition boundaries also appear to be smoother and the subdomains have a smaller maximum diameter (165, compared to 253 (KMETIS) and 179 (JOSTLE)). A similar trend w. r. t. the diameter can be found in an 8-partitioning



(a) KMETIS (b) JOSTLE (c) DIBAP-long

Figure 5.6.: Partitionings of the graph t60k ($|V| = 60005$, $|E| = 89440$) into $k = 12$ subdomains with the three partitioners.



(a) KMETIS (b) JOSTLE (c) DIBAP

Figure 5.7.: Partitionings of biplane9 ($|V| = 21701$, $|E| = 42038$) into $k = 8$ subdomains with the three partitioners.

of the smaller graph *biplane9*, see Figure 5.7. Also note the again smoother boundaries produced with DIBAP and that both other libraries generate a partition with two large disconnected node sets.

A detailed comparison of the diameter values for $k = 16$ on our benchmark graphs of Table 5.1 reveals similar results. DIBAP is on average 4.4% (ℓ_1-norm) and 5.9% (ℓ_∞-norm) better than JOSTLE, respectively. On KMETIS the improvements are slightly larger. Since disconnected subdomains (whose diameter is set to ∞) do not enter into these comparisons, the real values of KMETIS and JOSTLE tend to be worse than those computed and used for the comparison above. Our algorithm yields disconnected subdomains in only 2.1% of the experiments, while KMETIS exhibits a more than doubled ratio of 4.4%. Much worse is JOSTLE, which produces disconnected subdomains in 22.3% of the runs.

It must be noted that the gain in solution quality obtained with DIBAP versus KMETIS and JOSTLE is clearly bought with a higher computational effort. This becomes clear in Table 5.6, which shows the average running time in seconds for partitioning the benchmark set in different numbers of subdomains. The two established libraries are known to be very fast, and DIBAP is not able to keep up with their speed. To provide an average

Table 5.6.: Average running times in seconds on an Intel Core 2 Duo 6600 processor for partitioning the benchmark graphs with KMETIS, JOSTLE, and DIBAP ($\Lambda = 10, \phi = 14$).

k	KMETIS	JOSTLE	DIBAP
4	**0.33**	0.62	6.65
8	**0.34**	0.70	11.56
12	**0.35**	0.77	15.98
16	**0.36**	0.83	20.18
20	**0.37**	0.89	24.30
32	**0.39**	1.04	33.73
avg	**0.36**	0.81	18.73

value derived from the experiments with different numbers of subdomains, one can say that KMETIS is about 50 times faster than DIBAP, JOSTLE 20 times.

This speed gap becomes particularly evident for larger k, which is mainly due to the fact that – in contrast to KMETIS and JOSTLE – DIBAP scales nearly linearly with k, i. e., doubling k results in a nearly doubled running time. The linear scaling in k has already been observed with BUBBLE-FOS/C. There, it has been alleviated by partial graph coarsening. A similar idea might work here as well. If k is large (and each subdomain relatively small), the movement of the subdomains is likely to be rather small. Hence, a partial or adaptive coarsening of the active nodes depending on the distance to the boundary would reduce the number of necessary computations. Future work will need to show if this approach results in a similarly high solution quality.

On a closer look, however, the absolute running times of DIBAP are already quite satisfactory. They range from a few seconds to a few minutes for our benchmark graphs. Among other improvements, we plan for a distributed-memory parallelization of TRUNC-CONS. If one assumes a parallel graph partitioning or load balancing scenario with k processors for k partitions, one may divide the sequential running times of DIBAP by $k \cdot e$ (where $0 < e \leq 1$ denotes the efficiency of the parallel program). In such a case its parallel running time on k processors can be expected to be at most a few dozens of seconds even for large problems, which is certainly acceptable.

5.4.1.5. Influence of k and the Graph Size

Since BUBBLE-FOS/C apparently does not work well for $k = 2$, it is interesting to note that this flaw does not hold any more for DIBAP. Experiments analogous to those of the previous section reveal that DIBAP is in all four quality measures (external edges and boundary nodes in both norms) one to three percent better than KMETIS and JOSTLE. This is not as good as for higher k, probably because of the inferior starting solution provided by BUBBLE-FOS/C. Yet, it is much better than our previous algorithm and still improves on the competing libraries. This is all the more remarkable if one considers that these libraries are based on the KL/FM heuristic, which has been developed initially for bipartitioning.

Table 5.7.: Scaled running times of DibaP ($\Lambda = 10$, $\psi = 14$) on graphs from the *mrng* series with $k = 8$ (left) and on the benchmark graphs from Table 5.1 (right).

Graph	$\|V\|$	$\|E\|$	Time / $\|E\|$ (µs)
mrng1	257,000	505,048	19.07
mrng2	1,017,253	2,015,714	24.52
mrng3	4,039,160	8,016,848	15.63
mrng4	7,533,224	14,991,280	17.28

k	Time / k (s)	Time / $k^{0.8}$ (s)
4	1.66	2.19
8	1.45	2.19
12	1.33	2.19
16	1.26	2.2
20	1.22	2.21
32	1.05	2.11

To estimate how the graph size enters into the running time, an evaluation of DibaP's non-aggregated running times on the benchmark graph shows that $|E|$ is much more important than $|V|$. For example, the graph *ocean* can be partitioned faster than the graph *tooth*, although it has twice as many vertices. The reason for this is that it has fewer edges.

Additionally, we have conducted experiments on four graphs from the *mrng* series (dual graphs of 3D FEM meshes). These graphs have different sizes, but a similar structure. Thus, the results are not biased on the graph structure. The smallest graph *mrng1* has 257,000 nodes, the largest one *mrng4* around 7.5 million, see Table 5.7 (left). All four graphs have a very similar average degree of just below four. Without going into detail here, the experimental data confirm that the graph size enters approximately linearly into the running time. As a representative example, running times for $k = 8$ are shown in Table 5.7 (left). Note that the times are divided by $|E|$ and are given in microseconds. These values indicate that an increase in the graph size results in a very similar increase in running time. The primary reasons for variations in the data are the partition placement and the resulting number of (in)active nodes.

Table 5.7 (right) contains the average running times of DibaP divided by k on the eight benchmark graphs of Table 5.1. The division by the number of subdomains shows that the influence of k is not totally linear. If one divides the running time by $k^{0.8}$ (rightmost column), however, the results are nearly constant. While an extrapolation of these data to asymptotic behavior may be shaky, we can still conclude that our experiments mostly confirm our expectations drawn from theoretical considerations: DibaP scales *approximately* linearly with $|E|$ and k.

5.4.1.6. Best Known Edge-Cut Results

Chris Walshaw's benchmark archive [Wals 07b], from which the test graphs of this section have been taken, also collects the best known partitions for each of the 34 graphs contained therein. More precisely, it stores partitions with the lowest edge-cut currently known. At the moment results of more than 20 algorithms are considered. Many of these algorithms are significantly more time-consuming than METIS and Jostle used in our experiments above.

With each graph 24 partitions are recorded, one for six different numbers of subdomains ($k \in \{2, 4, 8, 16, 32, 64\}$) in four different imbalance settings ($0\%, 1\%, 3\%, 5\%$). Using DibaP in various parameter settings ($\Lambda \leq 15$, $\psi \leq 20$), we have been able to improve more than 80 of these currently best known edge-cut values for six of the eight largest graphs in the archive. More details can be found in Table A.11 in the appendix. The complete list of improvements with the actual edge-cut values and the corresponding partition files are available from Walshaw's archive.

Note that none of our records is for $k = 2$. We conjecture that this is the case because the starting solutions computed by Bubble-FOS/C are not really good for $k = 2$. Moreover, these records are mostly held by expensive tailor-made bipartitioning algorithms. Unless they are extended to $k > 2$, their high quality is not likely to sustain for larger k because recursive bipartitioning typically yields inferior results compared to direct k-way methods for large k [Simo 97].

5.4.2. Load Balancing by Repartitioning

5.4.2.1. Setting

The DibaP implementation used for the load balancing experiments includes the modifications discussed in Section 5.3. Otherwise, it shares the same code basis as the graph partitioning variant. Hence, unlike the parallel versions of METIS and Jostle, our load balancer is not prepared yet for a distributed-memory parallelization. That is why we concentrate in the following on the quality of the experiments and neglect their running time. Comparing the latter is part of future work once an MPI parallel version of DibaP exists. Major deviations from the running time ratios observed in the previous section on graph partitioning are not likely. On the other hand, we believe that DibaP's inherent parallelism is able to reduce the speed gap between our algorithm and the KL/FM partitioners somewhat.

In the previous section we have concluded that $\Lambda = 10$, $\psi = 14$, and three coarse solutions computed by Bubble-FOS/C are parameter settings that provide a very good trade-off between running time and quality. That is why we choose these values for all load balancing experiments. The other parameters for Bubble-FOS/C within DibaP are unchanged except for `thrsh`, which is set to $8,000$, and for the abdication of the virtual vertex notion (FOS/V).

Our benchmark set comprises two different sets of graph sequences. Twelve sequences are made of 101 frames of small graphs (around 10,000 to 15,000 nodes each), which are repartitioned into $k = 12$ subdomains. The second set consists of three sequences of larger graphs (between 110,000 and 1,100,000 nodes each), which are repartitioned into $k = 16$ subdomains. While the sequence *bigtric* has 101 frames, the sequences *bigbubbles* and *bigtrace* have only 46 frames. All graphs of these 15 sequences have a two-dimensional geometry and have been generated to resemble applications from numerical simulations such as fluid dynamics. (For more details on the generation process the reader is referred

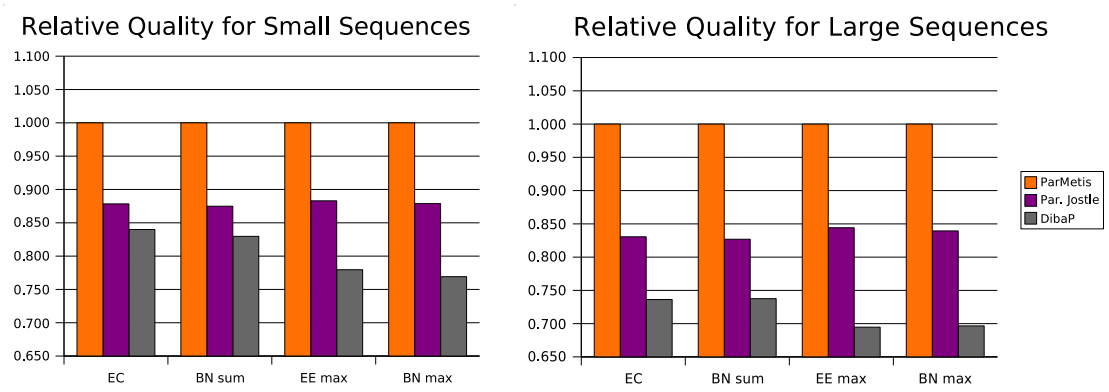


Figure 5.8.: Average partition quality (boundary nodes, external edges) in the ℓ_1- and ℓ_∞-norm for repartitionings relative to ParMETIS on twelve small (left) and three large (right) graph sequences.

to Marquardt and Schamberger [Marq 05], who have provided the sequence data. A visual impression of some of the data is given by Schamberger [Scha 06, p. 104f.]. The graph of frame $i+1$ in a given sequence is obtained from the graph of frame i by changes restricted to local areas. As an example, some areas are coarsened, whereas others are refined. These changes are in most cases due to the movement of an object in the simulation domain and often result in unbalanced subdomain sizes.

In addition to the graph partitioning metrics used in the previous section, we are here also interested in migration costs. These costs result from data that change their processor after the repartitioning process. As described in Section 1.3.2, we count the number of nodes that change their subdomain from one frame to the next as a measure of these costs. One could alternatively assign cost weights to the partition objective and the migration volume to evaluate the linear combination of both. Since these weights depend both on the underlying application and the parallel architecture, we have not pursued this further here.

One might wonder if a multilevel scheme is really needed for repartitioning. It could be possible that improvements on the finest graph already suffice. Our experiments indicate that a multilevel approach is indeed necessary in order to produce large enough movements of the subdomains that keep up with the movements of the simulation. Partitions generated by multilevel DibaP are of a noticeably higher quality regarding the graph partitioning metrics than by TruncCons without multilevel approach. Also, using a multilevel hierarchy results in very steady migration costs, which rarely deviate much from the mean. The partitioner ParMETIS seems to follow a different migration strategy. As we will see below, it tends to migrate either very many or very few vertices during a sequence. It is not easy to say which strategy is definitely better, but our experiments suggest clearly that ParMETIS yields higher migration costs than DibaP.

Table 5.8.: Average migration volume in the ℓ_1- and ℓ_∞-norm for repartitionings computed by ParMETIS, Jostle, and DibaP on all fifteen graph sequences.

	ParMETIS		Jostle		DibaP	
Sequence	mig_1	mig_∞	mig_1	mig_∞	mig_1	mig_∞
bubbles	2460.7	992.1	**1723.9**	623.3	1775.0	**586.0**
change	284.4	132.1	**330.9**	130.4	352.2	**119.8**
circles	3200.5	1226.9	**2128.6**	799.9	2164.1	**794.2**
fastrot	4314.3	1616.5	3229.6	**1211.9**	**3094.8**	1236.9
fasttric	4648.1	1756.9	3466.6	1313.0	**2940.0**	**1189.6**
heat	299.8	121.9	**286.5**	**106.9**	561.1	189.8
refine	**1.5**	**1.2**	114.2	37.2	30.6	10.6
ring	3369.8	1305.9	2684.0	765.4	**2584.1**	**692.2**
rotation	2914.9	1288.7	**2281.1**	1055.0	2421.6	**1028.4**
slowrot	3928.4	1451.8	2774.7	961.4	**2511.4**	**911.9**
slowtric	3094.9	1213.5	2322.1	**872.2**	**2165.1**	878.5
trace	977.5	388.1	896.0	327.7	**781.9**	**282.1**
bigtric	27563.2	8972.4	**20170.0**	6762.8	22248.3	**5938.8**
bigbubbles	197449.2	68469.0	**157475.0**	50356.1	182205.9	**46730.8**
bigtrace	71934.6	26166.9	**61294.1**	**20127.8**	90358.2	24898.8

5.4.2.2. Comparison to other Libraries

As before, we compare our new algorithm DibaP to the state-of-the-art. For repartitioning these are the parallel versions of the graph partitioners METIS and Jostle. The load balancing toolkit Zoltan [Cata 07], whose integrated KL/FM partitioner is based on the hypergraph concept, is not included in the detailed presentation. Our experiments with it indicate that its current version 3.0 is not as suitable for our benchmark set of FEM graphs. In the *repartition* setting Zoltan yields many disconnected subdomains and very high migration costs, while the number of external edges and boundary nodes are in general also higher than those of DibaP, ParMETIS, and Jostle. If set to *refine*, Zoltan migrates significantly less. At the same time the graph partitioning metrics become even worse. Insofar we conclude that currently the dedicated graph (as opposed to hypergraph) partitioners seem more suitable for this problem type.

The partitioning quality regarding the boundary nodes and external edges measured in our experiments with ParMETIS, Jostle, and DibaP are displayed in Figure 5.8, where the values are shown relative to ParMETIS. Moreover, the values are averaged over all small sequences on the left and over all large sequences on the right. (The corresponding non-aggregated data values can be found in Tables A.5, A.6, and A.7 in the appendix.) Additionally, we are interested in the migration costs, which are recorded in both norms and detailed for each sequence in Table 5.8.

The averaged graph partitioning metrics show that DibaP is able to compute the best partitions on average. DibaP's migration volume is best for six (out of 15) sequences in the summation norm. Compared to this, only parallel Jostle is competitive. Its par-

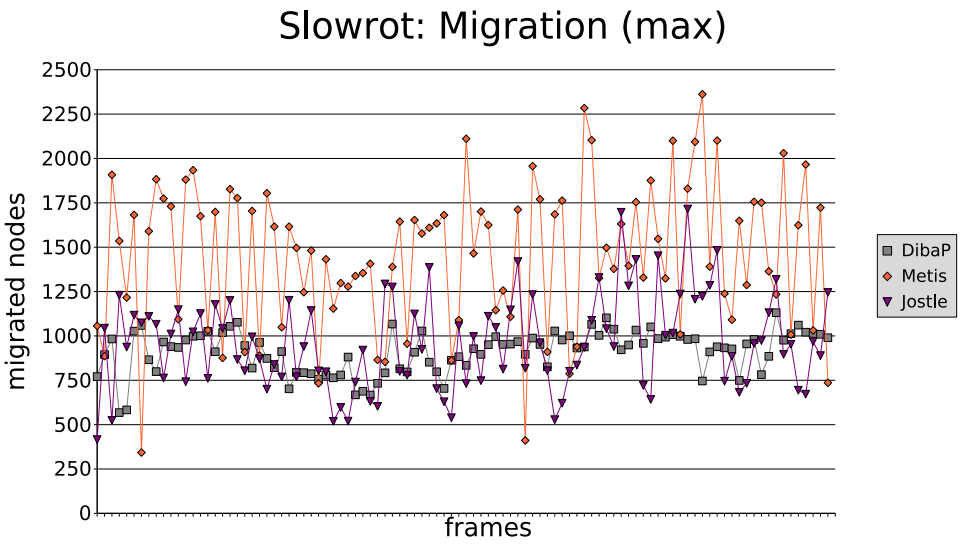


Figure 5.9.: Number of migrating nodes (ℓ_∞-norm) in each frame of the *slowrot* sequence for DibaP (grey square), METIS (orange diamond), and Jostle (purple triangle).

titions have a lower quality, but its migration volume is best for eight sequences in the summation norm. In the maximum norm the results are not fundamentally different, but DibaP performs even better than before. Again, it attains the best partitions. Moreover, the migration volume is also best for ten sequences. As a representative example, Figure 5.9 shows the migration volumes of each frame within the *slowrot* sequence in the ℓ_∞-norm. One can see the different strategies of the three programs. While Jostle and DibaP have a relatively constant migration volume, the values for ParMETIS fluctuate extremely. In general, these outcomes concerning the solution quality confirm results derived from experiments with Bubble-FOS/C [Meye 05]. The additional advantage of DibaP compared to Bubble-FOS/C is that the high solution quality can be exploited in a much more reasonable amount of time.

The load balancing results described above lead to the conclusion that our algorithm concentrates very much on getting good partitions. However, it seems to neglect the second objective migration costs in some cases. This behavior is not totally surprising since no explicit mechanisms for migration optimization are used within DibaP. Such mechanisms could be integrated if one finds in other experiments that DibaP's migration costs become too high. Reducing Λ, the number of consolidations, could already avoid large subdomain movements, depending on the input.

In summary one can say that, in almost all cases, DibaP computes the best repartitionings w. r. t. to the graph partitioning metrics. Concerning the migration volume, the results are not as clear. The ℓ_1-norm values are slightly in favor of Jostle (eight times best) compared to DibaP (six times best). Yet, in the ℓ_∞-norm of the migration volume, DibaP is the clear winner again. The strategy of ParMETIS to migrate either very few

or very many nodes does not seem to pay off on average since ParMETIS computes in most cases the worst solutions.

The graph partitioning experiments give rise to the conjecture that an MPI parallel version of DibaP will be significantly slower than ParMETIS and Jostle. On the other hand, input sizes with a few hundreds of thousands of nodes per processor can be expected to be repartitioned within a few dozens of seconds or minutes by DibaP, which is certainly acceptable is most cases. We would like to stress the fact that a high repartitioning quality is very important. Usually, the most time consuming parts of numerical simulations are the numerical solvers. Hence, a reduced communication volume provided by an excellent partition can be expected to pay off unless the repartitioning time is extremely high.

5.4.3. Graph Clustering

To analyze DibaP's graph clustering capabilities, we have constructed random graphs from two different models. Both follow the idea of the popular *planted partition* model devised by Jerrum and Sorkin [Jerr 98], which has been motivated already in Section 4.7.3. The first model we use is the same as the one used in Section 4.7.3 with Bubble-FOS/C, only the graphs are larger here. Our second choice is the original model of Jerrum and Sorkin, which is explained below. Besides DibaP, Bubble-FOS/C (this time with AMG hierarchy and solver for enhanced speed) is included in the second set of experiments to test it on another model as well. Both our algorithms do not use the smooth operation because it would worsen the results on these highly irregular instances.

Again, the kernel k-means implementation Graclus [Dhil 07] serves us as a standard of reference for evaluating our algorithms on graph clustering problems. Our choice is motivated by the following considerations. First of all, Graclus is freely available and has been designed for solving clustering problems that are also addressed by DibaP and Bubble-FOS/C. It also requires the specification of k and one of its optimization objectives is the normalized cut. Moreover, it is very fast, computes solutions of good quality, and can be regarded as state-of-the-art for optimizing the normalized cut criterion in our graph clustering problems. We forbear from the use of an algorithm based on the Euclidean Commute Time Distance. To compute these distances is very expensive due to the pseudoinversion of the Laplacian and hardly tractable for large instances.

The DibaP parameters that have been changed compared to the graph partitioning experiments are the number of Bubble operations and the switch threshold `thrsh`. The latter is set to $8,000$ for model 2; `thrsh` $= 4,000$ is about 30% faster, but yields slightly worse results. For model 1 we set `thrsh` to $2,000$ only. Higher values result in a lower solution quality. Consolidations within Bubble-FOS/C are not used. Thus, the number of `AssignPartition` and `ComputeCenters` operations is increased to 4 each.

Table 5.9.: Experimental results showing the normalized cut values computed by DibaP and Graclus for the graphs of model 1.

	Graclus			DibaP		
Graph size / k	6	8	12	6	8	12
2^{14}	**0.335**	**0.641**	**0.716**	0.363	**0.641**	0.803
2^{15}	0.337	**0.643**	**0.672**	**0.334**	**0.643**	0.703
2^{16}	0.339	**0.644**	0.716	**0.336**	**0.644**	**0.705**
2^{17}	0.338	**0.643**	**0.670**	**0.335**	**0.643**	0.955
2^{18}	-	**0.643**	0.671	-	**0.642**	**0.666**
average	**0.337**	**0.643**	**0.682**	0.342	**0.643**	0.757

5.4.3.1. Random Graph Model 1

The graphs of model 1 in this section have between 2^{14} and 2^{18} nodes. Their planted partitions have been constructed by normal distributions with the same parameters (except for the size) as their smaller counterparts in Section 4.7.3. Hence, the mean internal and external degree and their standard deviations are given by $\mu_{int} = 4.3, \sigma_{int} = 1.1, \mu_{ext} = 0.3, \sigma_{ext} = 0.3$ for $k = 6$ and $k = 12$ and $\mu_{int} = 5.1, \sigma_{int} = 1.3, \mu_{ext} = 0.45, \sigma_{ext} = 0.35$ for $k = 8$, resulting in node degrees between 1 and 12.

Table 5.9 shows the normalized cut values derived in the experiments on these graphs. For $k = 8$ both algorithms obtain the same values. Apparently, these instances are relatively easy, so that both algorithms always compute the same local optimum. Possibly, considering the different approaches of the algorithms, this local optimum might even be the global one. For $k = 6$ DibaP obtains the best values in three out of four cases, but Graclus is the best on average. Graclus also performs better for $k = 12$. In summary, one can say that the solution quality of both algorithms on these instances is similar. Yet, Graclus is slightly better, in particular for larger k, where DibaP sometimes inherits the Bubble-FOS/C problem of bad initial centers. Moreover, on average Graclus is by a factor of 34 faster than DibaP.

Note that the combination $n = 2^{18}$ and $k = 6$ is left out because the construction mechanism has failed repeatedly to build connected graphs. Graphs with more than one connected component are in principle no problem for the algorithm DibaP. Each component can be seen as a cluster, so that the actual work is spent on each component separately. However, since our algorithm optimizes the normalized cut only implicitly, an optimal combination of all components is not integrated into our implementation yet. That is why we use only connected graphs and leave the handling of disconnected ones to future work.

5.4.3.2. Random Graph Model 2

A graph $G = (V, E)$ of model 2 is generated differently than before. One also specifies the cluster sizes, but instead of drawing the node degrees from a normal distribution,

Table 5.10.: Experimental results showing the normalized cut values computed by Bubble-FOS/C, DibaP, and Graclus for the graphs of model 2.

	Graclus			Bubble-FOS/C			DibaP		
Setting / k	9	13	17	9	13	17	9	13	17
1	4.70	9.55	12.22	7.20	10.10	12.88	**4.32**	**8.23**	**12.03**
2	3.65	5.80	8.42	6.46	10.27	13.11	**3.31**	**5.46**	**7.80**
3	4.02	6.74	**9.57**	6.94	10.77	13.82	**3.44**	**5.84**	**9.57**
4	4.72	9.62	12.31	7.13	10.21	12.71	**4.60**	**8.15**	**12.11**
5	3.57	5.97	8.52	6.69	10.19	13.17	**2.82**	**5.33**	**7.75**
6	3.98	6.72	9.90	7.01	10.78	13.68	**3.67**	**6.19**	**9.80**
7	4.64	9.69	12.41	7.09	10.31	12.78	**4.30**	**8.01**	**12.07**
8	3.55	6.08	8.68	7.11	10.36	13.54	**2.99**	**5.24**	**8.14**
9	4.06	**6.88**	12.90	7.35	10.62	12.78	**3.57**	7.03	**9.53**
avg	4.10	7.45	10.55	7.00	10.40	13.16	**3.67**	**6.61**	**9.87**

one determines for each node pair (u, v) whether $\{u, v\} \in E$ based on the parameters p_{int} and p_{ext} in the following manner. Let $X_{u,v} \in [0, 1]$ be a random variable drawn from a standard uniform distribution for the pair (u, v). Then, if u and v are in the same cluster, $\{u, v\} \in E \Leftrightarrow X_{u,v} < p_{int}$. Similarly, if u and v are not in the same cluster, $\{u, v\} \in E \Leftrightarrow X_{u,v} < p_{ext}$.

For our experiments we have constructed 27 graphs in this way, nine for each number of clusters $k \in \{9, 13, 17\}$. Odd numbers have been selected for k to add more variation compared to previous experiments. The graph sizes range from $20,000$ over $40,000$ to $80,000$, which is not very large, so that we can include Bubble-FOS/C in our experiments as well. While the cluster sizes can vary considerably from the mean size, extremely large or small clusters are avoided because both diffusion-based algorithms have problems with such instances. Usually, the cluster sizes are at most three times larger or three times smaller than the mean. However, some clusters have only about 10% of the mean size. The parameters p_{int} and p_{ext} have been chosen inverse proportional to $n \cdot k$ to obtain a sparse structure. Their actual values are shown in Tables A.8, A.9, and A.10 in the appendix.

The evaluation of the normalized cut values obtained by Graclus, Bubble-FOS/C, and DibaP (Table 5.10) yields first of all that Bubble-FOS/C cannot compete with the other two programs. Insofar it is surprising that DibaP, the combination of Bubble-FOS/C and TruncCons, performs very well. It computes the best normalized cut values in 26 out of 27 cases. As a result, the average values are significantly better than those of Graclus. Based on these results, one can conclude that DibaP is more suitable for random graph instances of model 2 than the state-of-the-art program Graclus – at least if the cluster sizes do not deviate too much from the mean value. However, the better solution quality of DibaP has to be paid by a much higher running time. For the most time-consuming instance our algorithm requires 225s. Averaged over all instances in this model, Graclus is faster by a factor of 60. As indicated above, this factor can

be reduced below 50 by using a smaller value for `thrsh` without sacrificing the solution quality much.

5.5. Parallelism

Recall that the running times presented for DibaP are based on its POSIX threaded version. A comparison of the threaded implementation to its non-threaded counterpart reveals that the obtainable speedup is larger than that of Bubble-FOS/C. More precisely, the thread parallelization makes DibaP faster by a factor of 1.55 on average. This corresponds to an efficiency of 77.5% on the dual-core test machine. Such a value is not extremely good, but still satisfactory, considering the thread overhead and our program's sequential parts. In any case, it improves over Bubble-FOS/C, which achieves only a speedup of 1.3 on average.

In a future distributed-memory implementation of DibaP, the major concern should lie on an efficient execution of TruncCons. Assuming that the graph is distributed over the processors, the communication of the updated load values in each FOS/T iteration can be very fine-granular. Such a low ratio of computational operations versus communication operations may prevent high speedups, depending on the parallel machine architecture employed. To circumvent this problem, one could use the basic idea of domain sharing presented in Section 4.8.1. Before computing the load values, each processor determines for subdomain i, $0 \leq i < k$, the possibly active nodes. These are those nodes whose distance to the subdomain boundary is at most ψ. Then, the subgraph induced by these nodes is sent to processor i. This processor combines all received subgraphs. It can then compute the FOS/T diffusion loads without further communication except for sending back the final load values to the originating processors. The Bubble-FOS/C experiments described in Section 4.8.2 for domain sharing and partial graph coarsening show that two large communication operations are often faster than many small ones.

A complex parallelization of Bubble-FOS/C with its AMG solver might not be necessary. Instead, each processor computes one (or more) initial Bubble-FOS/C solution(s) on the coarsest TruncCons level. The best one of these solutions is communicated to all processors and constitutes the starting point for the multilevel TruncCons improvement process. Since Bubble-FOS/C is quite fast on small graphs, the running time of such a concurrent sampling for the initial solution can be expected to be marginal.

5.6. Discussion

The local improvement scheme TruncCons developed in this chapter is a fast and very effective tool for improving graph partitions. Our new algorithm DibaP, which combines Bubble-FOS/C and TruncCons, attains excellent results in graph partitioning, as our experiments on popular benchmark graphs show. It achieves partitions with a significantly higher quality than kMetis and Jostle, two very popular state-of-the-art

libraries. While shorter boundary lengths have also been attained with Bubble-FOS/C, DibaP computes fewer external edges in the summation *and* the maximum norm, too. This is confirmed by the computation of a large number of best-known partitions w. r. t. the edge-cut for six graphs of a popular benchmark archive.

There are two possible reasons for the improved edge-cut results of the hybrid approach DibaP compared to using Bubble-FOS/C alone. First and foremost, in Section 3.2 we have argued that nodes of the same cluster are connected by many shortest paths of small length, whereas the shortest paths of nodes in different clusters lead via very few external edges. Since the local improvement scheme TruncCons uses exactly this notion of random walks of short length (whereas Bubble-FOS/C considers random walks of all lengths), it is able to detect which nodes and regions are connected to each other by these short paths and which are not. Thereby TruncCons improves the boundary regions of reasonable initial partitions very well w. r. t. the edge-cut. The second – probably less important – reason could be the different coarsening scheme. While the employed matching algorithm aims at a very uniform coarsening, the AMG coarsening algorithm produces few nodes with large degree and many nodes of small degree. For large graphs this irregularity in the AMG coarsening might lead to somewhat worse solutions because the structure of the original graph is not retained well enough on the coarse levels.

Another advantage of DibaP compared to the KL/FM heuristic is that the computed subdomains appear to be more compact with smoother boundaries than those computed by kMetis and Jostle. Although DibaP is clearly slower than the two established KL/FM libraries, its running time on a dual-core processor is acceptable for all but extremely large inputs. Even a graph with 7.5 million nodes and 15 million edges can be partitioned into eight subdomains within five minutes on commodity dual-core hardware. In particular, DibaP is much faster than our previous algorithm Bubble-FOS/C.

The load balancing quality of DibaP is very good due to the superior partitions computed. The required migration volume is not in all cases better than with state-of-the-art libraries, but the experiments reveal an advantage at least for the maximum norm. In particular for problems that favor a high partition quality over migration volume and duration of repartitioning, we advocate DibaP as the tool of choice.

Clustering problems on random graphs are solved by DibaP with a high solution quality in many cases. While random graphs of one model are clustered slightly better by Graclus, DibaP is clearly the best for the widely used second planted partition model. Yet, the graphs are connected and cluster sizes in the experimental data are not chosen completely arbitrarily. The extension of our results to extremely small or large clusters should be part of future work. One possible approach could be to start with a very large number of clusters and to merge them iteratively. The clusters to merge are chosen greedily based on the improvement of the optimization objective. Another possible improvement is the integration of local search techniques. They might help to avoid problems with bad solutions on the coarsest level, in particular when k is large.

Although DibaP offers a superior solution quality than the state-of-the-art in many cases, its deployment in time-critical applications also depends on its running time. Currently, the speed gap between our algorithm and its best competitors is between one and two orders of magnitude. This is certainly a significant difference, but there are several starting points to improve the situation in future work. One of them consists in eliminating the nearly linear dependence on k in the running time. This might be possible by techniques similar to partial graph coarsening, which has been partially effective for Bubble-FOS/C.

Another aspect for future work is a distributed-memory parallelization. Since the most time-consuming parts of DibaP (the diffusive operations within TruncCons) exhibit a large degree of parallelism, significant accelerations can be expected. An efficient parallelization might also close the speed gap partially because the KL/FM heuristic within METIS and Jostle and the local search of Graclus are difficult to parallelize. Furthermore, the simplicity of the diffusive operations within TruncCons makes the use of fast parallel streaming hardware possible. As part of future work, we investigate the explicit use of SIMD instruction extensions of modern CPUs and the use of fast streaming graphics processors. Since graphics processors have a much higher performance for such simple operations than CPUs, additional accelerations might be possible.

6. Conclusions and Future Work

Three related problems have been considered in this thesis, graph partitioning, load balancing by repartitioning, and graph clustering. All of them have in common that they are combinatorially difficult and require the identification of densely connected regions of a graph. In the introduction we have argued that heuristics which determine subdomains with good shapes are very promising, in particular for graph partitioning and repartitioning. Drawbacks of previous shape-optimizing techniques have been identified and then eliminated by the introduction of our new similarity measure FOS/C, which is based on disturbed diffusion. After its theoretical analysis, which involves its relation to random walks, we use FOS/C as a similarity measure in the partitioning algorithm Bubble-FOS/C.

By introducing algebraic multigrid techniques into the solution process of Bubble-FOS/C, we have accelerated the algorithm significantly without sacrificing its high solution quality for graph partitioning. However, the running time of Bubble-FOS/C and its quality for clustering problems are still not competitive to the state-of-the-art in our experiments. Thus, we have reiterated the main core of the algorithm engineering cycle [Sand 07], which consists of design, analysis, implementation, and experimental evaluation. As a result, we have devised the much faster algorithm DibaP, which has a running time that is approximately linear in $|E|$ and k and attains a very high quality in all considered applications. For graph partitioning it computes a significant number (more than 80 out of 144) of best known edge-cut values for six of the eight largest graphs contained in a well-known benchmark set. Additionally, extensive experiments demonstrate that DibaP delivers better partitions than kMetis and Jostle – two state-of-the-art graph partitioning libraries using the KL/FM heuristic. The outcomes of the repartitioning and graph clustering experiments are not as clear, but they also show that DibaP's solution quality competes with the state-of-the-art or is even better.

All these results verify our introductory assumption that diffusive shape optimization is a successful approach for providing (re)partitions of superior quality without requiring geometric information. Our algorithm DibaP overcomes the drawbacks of traditional KL-based algorithms, so that it meets the requirements most users expect from a graph (re)partitioner. Moreover, its linear running time improves on many related graph clustering algorithms. It must be noted, however, that the speed gap to fast state-of-the-art tools in partitioning and clustering is still between one and two orders of magnitude, although the absolute running times of DibaP are quite satisfactory.

Future work should therefore concentrate on a further acceleration of diffusive partitioning techniques. Of utmost importance is a distributed-memory parallelization of DibaP, which would in the ideal case also eliminate the nearly linear dependence on k in the running time. Theoretically, starting from our convergence results, it would be interesting to obtain more knowledge on the relation of the Bubble framework and disturbed diffusion schemes. Of particular concern is the behavior of TruncCons and how to guarantee connected partitions. Another interesting aspect is an extension of our results to directed graphs. Since their Laplacian and diffusion matrices are not symmetric, many techniques from linear algebra we have used before are not applicable.

Considering that heterogeneous and hierarchical computing environments have become common, the generation of partitions specifically suited for simulations executed on such processor topologies would be an interesting extension. Hierarchical techniques could also help in the design of disturbed diffusive clustering methods that do not require the number of clusters a priori. An elimination of the parameter k could be possibly done by a multilevel algorithm in the spirit of DibaP, which combines two clustering algorithms. The coarsest solution would be computed by an (expensive) algorithm that determines k; the multilevel refinement could be done by a faster algorithm such as TruncCons. Tests will need to show if such a multilevel procedure is successful. Other possible extensions for graph clustering include the extension to time-dependent graphs that do not allow global knowledge and a higher robustness of the clustering quality for instances with completely arbitrary cluster sizes.

Bibliography

[Adam 91] J. Adamek. *Foundations of Coding: Theory and Applications of Error-Correcting Codes with an Introduction to Cryptography and Information Theory.* John Wiley & Sons, Inc., 1991.

[Alon 00] N. Alon and J. H. Spencer. *The Probabilistic Method.* J. Wiley & Sons, 2nd Ed., 2000.

[Aror 04] S. Arora, E. Hazan, and S. Kale. "$0(\sqrt{\log n})$ Approximation to SPARSEST CUT in $\widetilde{O}(n^2)$ Time". In: *Proceedings of the 45th Symposium on Foundations of Computer Science (FOCS'04)*, pp. 238–247, IEEE Computer Society, 2004.

[Aror 07] S. Arora and S. Kale. "A combinatorial, primal-dual approach to semidefinite programs". In: *Proceedings of the 39th Annual ACM Symposium on Theory of Computing (STOC'07)*, pp. 227–236, ACM, 2007.

[Baño 06] R. Baños, C. Gil, B. Paechter, and J. Ortega. "Parallelization of population-based multi-objective meta-heuristics: An empirical study". *Applied Mathematical Modelling*, Vol. 30, No. 7, pp. 578–592, July 2006.

[Berg 97] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational geometry: algorithms and applications.* Springer-Verlag, 1997.

[Bich 07] C.-E. Bichot. "A new Method, the Fusion Fission, for the relaxed k-way graph partitioning problem, and comparisons with some Multilevel algorithms". *Journal of Mathematical Modelling and Algorithms*, Vol. 6, No. 3, pp. 319–344, 2007.

[Bigg 93] N. Biggs. *Algebraic Graph Theory.* Cambridge University Press, 1993.

[Boil 90] J. E. Boillat. "Load Balancing and Poisson Equation in a Graph". *Concurrency – Practice & Experience*, Vol. 2, No. 4, pp. 289–314, 1990.

[Boll 98] B. Bollobás. *Modern Graph Theory.* Springer-Verlag, 1998.

[Bon 07] J. van Bon. "Finite primitive distance-transitive graphs". *European Journal of Combinatorics*, Vol. 28, No. 2, pp. 517–532, February 2007.

[Bran 07] U. Brandes, M. Gaertler, and D. Wagner. "Engineering Graph Clustering: Models and Experimental Evaluation". *ACM Journal of Experimental Algorithmics*, Vol. 12, 2007. Article 1.1.

[Brig 00] W. L. Briggs, V. E. Henson, and S. F. McCormick. *A multigrid tutorial.* SIAM, 2nd Ed., 2000.

[Bron 97] I. N. Bronstein, K. A. Semendjajew, G. Musiol, and H. Mühlig. *Taschenbuch der Mathematik.* Verlag Harri Deutsch, 3rd Ed., 1997.

[Cata 01] U. Catalyurek and C. Aykanat. "A hypergraph-partitioning approach for coarse-grain decomposition". In: *Proceedings of the 2001 ACM/IEEE Conference on Supercomputing*, p. 28 (CD), ACM, 2001.

[Cata 07] U. Catalyurek, E. Boman, K. Devine, D. Bozdag, R. Heaphy, and L. Riesen. "Hypergraph-based Dynamic Load Balancing for Adaptive Scientific Computations". In: *Proceedings of the 21st International Parallel and Distributed Processing Symposium (IPDPS'07)*, IEEE Computer Society, 2007. Best Algorithms Paper Award.

[Char 07] P. Chardaire, M. Barake, and G. P. McKeown. "A PROBE-Based Heuristic for Graph Partitioning". *IEEE Transactions Comput.*, Vol. 56, No. 12, pp. 1707–1720, 2007.

[Chev 06] C. Chevalier and F. Pellegrini. "Improvement of the Efficiency of Genetic Algorithms for Scalable Parallel Graph Partitioning in a Multi-level Framework". In: *Proceedings of the 12th International Euro-Par Conference*, pp. 243–252, Springer-Verlag, 2006.

[Chun 97] F. R. K. Chung. *Spectral Graph Theory (CBMS Regional Conference Series in Mathematics, No. 92)*. American Mathematical Society, February 1997.

[Come 06] F. Comellas and E. Sapena. "A Multiagent Algorithm for Graph Partitioning". In: *Applications of Evolutionary Computing, Proceedings of EvoWorkshops 2006*, pp. 279–285, Springer-Verlag, 2006.

[Corm 01] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2nd Ed., 2001.

[Cybe 89] G. Cybenko. "Dynamic Load Balancing for Distributed Memory Multiprocessors". *Parallel and Distributed Computing*, Vol. 7, pp. 279–301, 1989.

[Dell 06] M. Dellnitz, M. Hessel-von Molo, P. Metzner, R. Preis, and C. Schütte. "Graph algorithms for dynamical systems". In: *Modeling and Simulation of Multiscale Problems*, pp. 619–646, Springer-Verlag, 2006.

[Demm 97] J. W. Demmel. *Applied numerical linear algebra*. SIAM, 1997.

[Devi 06] K. D. Devine, E. G. Boman, R. T. Heaphy, R. H. Bisseling, and U. V. Catalyurek. "Parallel Hypergraph Partitioning for Scientific Computing". In: *Proceedings of the 20th International Parallel and Distributed Processing Symposium (IPDPS'06)*, IEEE, 2006.

[Dhil 04] I. Dhillon, Y. Guan, and B. Kulis. "A Unified View of Kernel k-means, Spectral Clustering and Graph". Tech. Rep. TR-04-25, University of Texas at Austin, Department of Computer Science, 2004.

[Dhil 07] I. S. Dhillon, Y. Guan, and B. Kulis. "Weighted Graph Cuts without Eigenvectors: A Multilevel Approach". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 29, No. 11, pp. 1944–1957, 2007.

[Diek 00] R. Diekmann, R. Preis, F. Schlimbach, and C. Walshaw. "Shape-optimized Mesh Partitioning and Load Balancing for Parallel Adaptive FEM". *J. Parallel Computing*, Vol. 26, pp. 1555–1581, 2000.

[Diek 95] R. Diekmann, B. Monien, and R. Preis. "Using Helpful Sets to Improve Graph Bisections". In: D. F. Hsu, A. L. Rosenberg, and D. Sotteau, Eds., *Interconnection Networks and Mapping and Scheduling Parallel Computations*, pp. 57–73, AMS, 1995.

[Diek 99] R. Diekmann, A. Frommer, and B. Monien. "Efficient schemes for nearest neighbor load balancing". *Parallel Computing*, Vol. 25, No. 7, pp. 789–812, 1999.

[Ding 04] C. Ding and X. He. "K-means Clustering via Pricipal Component Analysis". In: *Proceedings of the 21st International Conference on Machine Learning*, pp. 225–232, ACM, 2004.

[Dong 00] S. van Dongen. *Graph Clustering by Flow Simulation.* PhD thesis, Univ. of Utrecht, 2000.

[Doyl 84] P. G. Doyle and J. L. Snell. *Random Walks and Electric Networks.* Math. Assoc. of America, 1984.

[Drak 05] D. E. Drake Vinkemeier and S. Hougardy. "A linear-time approximation algorithm for weighted matchings in graphs". *ACM Transactions Algorithms*, Vol. 1, No. 1, pp. 107–122, 2005.

[Drie 95] R. V. Driessche and D. Roose. "A Graph Contraction Algorithm for the Fast Calculation of the Fiedler Vector of a Graph". In: *Proceedings of the 7th Conference Parallel Processing for Scientific Computing (PPSC'95)*, pp. 621–626, SIAM, 1995.

[Drin 04] P. Drineas, A. Frieze, R. Kannan, S. Vempala, and V. Vinay. "Clustering Large Graphs via the Singular Value Decomposition". *Mach. Learn.*, Vol. 56, No. 1-3, pp. 9–33, 2004.

[Elli 01a] R. B. Ellis. "Discrete Green's functions for products of regular graphs". In: *AMS National Conference, invited talk, special session on Graph Theory*, 2001.

[Elli 01b] R. B. Ellis. "Torus Hitting Times and Green's Functions". `http://math.iit.edu/~rellis/comb/torus/torus.html`, 2001. Last access: 24 Jan 2008.

[Elsn 05] U. Elsner. "The influence of random number generators on graph partitioning algorithms". *Electr. Transactions on Numerical Analysis*, Vol. 21, pp. 125–133, 2005.

[Enri 02] A. J. Enright, S. van Dongen, and C. A. Ouzounis. "An efficient algorithm for large-scale detection of protein families". *Nucleic Acids Research*, Vol. 30, No. 7, pp. 1575–1584, 2002.

[Fidu 82] C. M. Fiduccia and R. M. Mattheyses. "A linear-time heuristic for improving network partitions". In: *Proceedings of the 19th Conference on Design automation (DAC'82)*, pp. 175–181, IEEE Press, 1982.

[Fied 73] M. Fiedler. "Algebraic connectivity of graphs". *Czechoslovak Mathematical Journal*, Vol. 23, No. 98, pp. 298–305, 1973.

[Fied 75] M. Fiedler. "A property of eigenvectors of nonnegative symmetric matrices and its application to graph theory". *Czechoslovak Mathematical Journal*, Vol. 25, pp. 619–633, 1975.

[Flak 02] G. W. Flake, R. Tarjan, and K. Tsioutsiouliklis. "Graph Clustering and Minimum Cut Trees". *Internet Mathematics*, Vol. 1, No. 4, pp. 385–408, 2002.

[Fous 07] F. Fouss, A. Pirotte, J.-M. Renders, and M. Saerens. "Random-Walk Computation of Similarities between Nodes of a Graph with Application to Collaborative Recommendation". *IEEE Transactions on Knowledge and Data Engineering*, Vol. 19, No. 3, pp. 355–369, 2007.

[Fox 94] G. Fox, R. Williams, and P. Messina. *Parallel Computing Works!* Morgan Kaufmann, 1994.

[Frie 07] S. Friedhoff and M. Heming. "Laplace-Matrizen Iterationsverfahren". Bachelor Thesis, Department of Mathematics and Informatics, Bergische Universität Wuppertal, 2007.

[Frit 08] D. Fritzsche, V. Mehrmann, D. B. Szyld, and E. Virnik. "An SVD approach to identifying meta-stable states of Markov chains". *Electronic Transactions on Numerical Analysis*, Vol. 29, pp. 46–69, 2008.

[Gaer 05] M. Gaertler. "Clustering". In: U. Brandes and T. Erlebach, Eds., *Network Analysis: Methodological Foundations*, pp. 178–215, Springer-Verlag, 2005.

[Gare 74] M. R. Garey, D. S. Johnson, and L. Stockmeyer. "Some simplified NP-complete problems". In: *Proceedings of the 6th Annual ACM Symposium on Theory of Computing (STOC'74)*, pp. 47–63, ACM Press, 1974.

[Gods 01] C. Godsil and G. Royle. *Algebraic Graph Theory.* Springer-Verlag, April 2001.

[Golu 96] G. H. Golub and C. F. V. Loan. *Matrix Computations.* Johns Hopkins Univ. Press, 3rd Ed., 1996.

[Grim 01] G. R. Grimmett and D. R. Stirzaker. *Probability and Random Processes.* Oxford University Press, 3rd Ed., 2001.

[Gros 04] J. L. Gross and J. Yellen, Eds. *Handbook of Graph Theory.* CRC Press, 2004.

[Hare 01] D. Harel and Y. Koren. "On Clustering Using Random Walks". In: *Proceedings of 21st Foundations of Software Technology and Theoretical Computer Science (FSTTCS'01)*, pp. 18–41, Springer-Verlag, 2001.

[Hend 94] B. Hendrickson and R. Leland. *The Chaco user's guide – Version 2.0.* 1994.

[Hend 95a] B. Hendrickson and R. Leland. "A Multi-Level Algorithm For Partitioning Graphs". In: *Proceedings Supercomputing '95*, p. 28 (CD), ACM Press, 1995.

[Hend 95b] B. Hendrickson and R. Leland. "An improved spectral graph partitioning algorithm for mapping parallel computations". *SIAM J. Sci. Comput.*, Vol. 16, No. 2, pp. 452–469, 1995.

[Hend 96] B. Hendrickson, R. Leland, and R. V. Driessche. "Enhancing Data Locality by Using Terminal Propagation". In: *Proceedings of the 29th Hawaii International Conference on System Sciences (HICSS'96) Volume 1: Software Technology and Architecture*, p. 565, IEEE Computer Society, 1996.

[Hend 98] B. Hendrickson. "Graph Partitioning and Parallel Solvers: Has the Emperor No Clothes?". In: *Proceedings of Irregular'98*, pp. 218–225, Springer-Verlag, 1998.

[Hens 02] V. E. Henson and U. Meier-Yang. "BoomerAMG: A parallel algebraic multigrid solver and preconditioner". *Appl. Numer. Math.*, Vol. 41, No. 1, pp. 155–177, 2002.

[Hrom 91] J. Hromkovič and B. Monien. "The Bisection Problem for Graphs of Degree 4 (Configuring Transputer Systems)". In: *Proceedings of the 16th International Symposium on Mathematical Foundations of Computer Science (MFCS'91)*, pp. 211–220, 1991.

[Hu 99] Y. F. Hu and R. F. Blake. "An Improved Diffusion Algorithm for Dynamic Load Balancing". *Parallel Computing*, Vol. 25, No. 4, pp. 417–444, 1999.

[Huan 06] S. Huang, E. Aubanel, and V. C. Bhavsar. "PaGrid: A Mesh Partitioner for Computational Grids". *J. Grid Comput.*, Vol. 4, No. 1, pp. 71–88, 2006.

[Jain 99] A. K. Jain, M. N. Murty, and P. J. Flynn. "Data clustering: a review". *ACM Computing Surveys*, Vol. 31, No. 3, pp. 264–323, 1999.

[JaJa 92] J. JáJá. *An Introduction to Parallel Algorithms.* Addison-Wesley, 1992.

[Jerr 98] M. Jerrum and G. B. Sorkin. "The metropolis algorithm for graph bisection". *Discrete Appl. Math.*, Vol. 82, No. 1-3, pp. 155–175, 1998.

[Kaas 88] E. F. Kaasschieter. "Preconditioned conjugate gradients for solving singular systems". *J. of Computational and Applied Mathematics*, Vol. 24, No. 1-2, pp. 265–275, 1988.

[Kaib 04] V. Kaibel. "On the Expansion of Graphs of 0/1-Polytopes". In: M. Grötschel, Ed., *The Sharpest Cut: The Impact of Manfred Padberg and His Work*, pp. 199–216, SIAM, 2004.

[Kann 04] R. Kannan, S. Vempala, and A. Vetta. "On Clusterings: Good, Bad and Spectral". *Journal of the ACM*, Vol. 51, No. 3, pp. 497–515, 2004.

[Kary 98a] G. Karypis and V. Kumar. *MeTiS: A Software Package for Partitioning Unstrctured Graphs, Partitioning Meshes, [...], Version 4.0.* 1998.

[Kary 98b] G. Karypis and V. Kumar. "Multilevel k-way Partitioning Scheme for Irregular Graphs". *J. Parallel Distrib. Comput.*, Vol. 48, No. 1, pp. 96–129, 1998.

[Kauf 96] H. Kaufmann and H. Pape. "Clusteranalyse". In: L. Fahrmeir, A. Hamerle, and G. Tutz, Eds., *Multivariate statistische Verfahren*, Walter de Gryter & Co., 1996.

[Kern 70] B. W. Kernighan and S. Lin. "An efficient heuristic for partitioning graphs". *Bell Systems Technical Journal*, Vol. 49, pp. 291–308, 1970.

[Khan 06] R. Khandekar, S. Rao, and U. Vazirani. "Graph Partitioning using Single Commodity Flows". In: *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC'06)*, pp. 385–390, ACM, 2006.

[Koro 04] P. Korosec, J. Silc, and B. Robic. "Solving the mesh-partitioning problem with an ant-colony algorithm". *Parallel Computing*, Vol. 30, No. 5-6, pp. 785–801, 2004.

[Lafo 06] S. Lafon and A. B. Lee. "Diffusion Maps and Coarse-Graining: A Unified Framework for Dimensionality Reduction, Graph Partioning and Data Set Parametrization". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 28, No. 9, pp. 1393–1403, 2006.

[Lang 04] K. Lang and S. Rao. "A Flow-Based Method for Improving the Expansion or Conductance of Graph Cuts". In: *Proceedings of the 10th International Conference on Integer Programming and Combinatorial Optimization (IPCO'04)*, pp. 325–337, Springer-Verlag, 2004.

[Lang 05] K. Lang. "Fixing two weaknesses of the Spectral Method". In: *Proceedings of Advances in Neural Information Processing Systems 18 (NIPS'05)*, 2005.

[Leig 92] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*. Morgan Kaufmann Publishers, 1992.

[Lloy 82] S. P. Lloyd. "Least squares quantization in PCM". *IEEE Transactions on Information Theory*, Vol. 28, No. 2, pp. 129–136, 1982.

[Lova 93] L. Lovász. "Random Walks on Graphs: A Survey". *Combinatorics, Paul Erdös is Eighty*, Vol. 2, pp. 1–46, 1993.

[Mang 66] M. Mangad. "Bounds for the Two-Dimensional Discrete Harmonic Green's Function". *Mathematics of Computation*, Vol. 20, No. 93, pp. 60–67, Jan. 1966.

[Marq 05] O. Marquardt and S. Schamberger. "Open Benchmarks for Load Balancing Heuristics in Parallel Adaptive Finite Element Computations". In: *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, (PDPTA'05)*, pp. 685–691, CSREA Press, 2005.

[Maue 07] J. Maue and P. Sanders. "Engineering Algorithms for Approximate Weighted Matching". In: *Proceedings of the 6th International Workshop on Experimental Algorithms (WEA'07)*, pp. 242–255, Springer-Verlag, 2007.

[McDo 95] J. J. McDonald, M. Neumann, H. Schneider, and M. J. Tsatsomeros. "Inverse M-Matrix Inequalities and Generalized Ultrametric Matrices". *Linear Algebra and Its Applications*, Vol. 220, pp. 321–341, Apr. 1995.

[Meis 05] A. Meister. *Numerik linearer Gleichungssysteme*. Vieweg, 2nd Ed., 2005.

[Meye 05] H. Meyerhenke and S. Schamberger. "Balancing Parallel Adaptive FEM Computations by Solving Systems of Linear Equations". In: *Proceedings of the 11th International Euro-Par Conference*, pp. 209–219, Springer-Verlag, 2005.

[Meye 06a] H. Meyerhenke, B. Monien, and S. Schamberger. "Accelerating Shape Optimizing Load Balancing for Parallel FEM Simulations by Algebraic Multigrid". In: *Proceedings of the 20th IEEE International Parallel and Distributed Processing Symposium (IPDPS'06)*, p. 57 (CD), IEEE Computer Society, 2006.

[Meye 06b] H. Meyerhenke and T. Sauerwald. "Analyzing Disturbed Diffusion on Networks". In: *Proceedings of the 17th International Symposium on Algorithms and Computation (ISAAC'06)*, pp. 429–438, Springer-Verlag, 2006.

[Meye 06c] H. Meyerhenke and S. Schamberger. "A Parallel Shape Optimizing Load Balancer". In: *Proceedings of the 12th International Euro-Par Conference*, pp. 232–242, Springer-Verlag, 2006.

[Meye 07] H. Meyerhenke, B. Monien, S. Schamberger, and T. Sauerwald. "Graph Clustering based on Disturbed Diffusion". In: *Proceedings of the Oberwolfach Workshop Algorithm Engineering, Math. Forschungsinstitut Oberwolfach Report No. 25/2007*, pp. 1430–1431, 2007.

[Meye 08] H. Meyerhenke, B. Monien, and T. Sauerwald. "A New Diffusion-based Multilevel Algorithm for Computing Graph Partitions of Very High Quality". In: *Proceedings of the 22nd International Parallel and Distributed Processing Symposium (IPDPS'08) (to appear)*, IEEE Computer Society, 2008. Best Algorithms Paper Award.

[Moni 00] B. Monien, R. Preis, and R. Diekmann. "Quality Matching and Local Improvement for Multilevel Graph-Partitioning". *Parallel Computing*, Vol. 26, No. 12, pp. 1609–1634, 2000.

[Moni 04] B. Monien and S. Schamberger. "Graph Partitioning with the Party Library: HelpfulSets in Practice". In: *Proceedings of the 16th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'04)*, pp. 198–205, IEEE Computer Society, 2004.

[Moni 06] B. Monien, S. Schamberger, U.-P. Schroeder, and H. Meyerhenke. "On Balancing of Dynamic Networks". In: *New Trends in Parallel & Distributed Computing, Proceedings of the 6th International Heinz Nixdorf Symposium*, pp. 171–181, HNI Verlagsschriftenreihe, 2006.

[Moni 07] B. Monien, R. Preis, and S. Schamberger. "Approximation Algorithms for Multilevel Graph Partitioning". In: T. F. Gonzalez, Ed., *Handbook of Approximation Algorithms and Metaheuristics*, Chap. 60, pp. 1–15, Taylor & Francis, 2007.

[Muth 98] S. Muthukrishnan, B. Ghosh, and M. H. Schultz. "First- and Second-Order Diffusive Methods for Rapid, Coarse, Distributed Load Balancing". *Theory Comput. Syst.*, Vol. 31, pp. 331–354, 1998.

[Newm 04] M. E. J. Newman and M. Girvan. "Finding and evaluating community structure in networks". *Physical Review E (Statistical, Nonlinear, and Soft Matter Physics)*, Vol. 69, No. 2, 2004.

[Newm 06] M. E. J. Newman. "Modularity and community structure in networks". *Proceedings of National Academy of Sciences*, Vol. 103, p. 8577, 2006.

[Newm 82] D. J. Newman. "The hexagon theorem". *IEEE Transactions on Information Theory*, Vol. 28, No. 2, pp. 137–138, 1982.

[Ng 01] A. Y. Ng, M. I. Jordan, and Y. Weiss. "On spectral clustering: Analysis and an algorithm". In: *Proceedings of Advances in Neural Information Processing Systems 14 (NIPS'01)*, pp. 849–856, 2001.

[Norr 97] J. R. Norris. *Markov Chains.* Cambridge University Press, 1997.

[Olik 98] L. Oliker and R. Biswas. "PLUM: Parallel Load Balancing for Adaptive Unstructured Meshes". *J. Parallel and Distributed Computing*, Vol. 52, No. 2, pp. 150–177, 1998.

[Page 74] R. L. Page. "ACM Algorithm 479: A minimal spanning tree clustering method". *Commun. ACM*, Vol. 17, No. 6, pp. 321–323, 1974.

[Pell 07a] F. Pellegrini. "A parallelisable multi-level banded diffusion scheme for computing balanced partitions with smooth boundaries". In: *Proceedings of the 13th International Euro-Par Conference*, pp. 195–204, Springer-Verlag, 2007.

[Pell 07b] F. Pellegrini. "Scotch and libScotch 5.0 User's Guide". Tech. Rep., LaBRI, Université Bordeaux I, December 2007.

[Poth 90] A. Pothen, H. Simon, and K. Liou. "Partitioning sparse matrices with eigenvectors of graphs". *SIAM Journal of Matrix Analysis*, Vol. 11, pp. 430–452, 1990.

[Prei 99] R. Preis. "Linear time 1/2-approximation algorithm for maximum weighted matching in general graphs". In: *Proceedings of the 16th Symposium on Theoretical Aspects of Computer Science (STACS'99)*, pp. 259–269, Springer-Verlag, 1999.

[Puu 05] T. Puu. "On the Genesis of Hexagonal Shapes". *Networks and Spatial Economics*, Vol. 5, No. 1, pp. 5–20, March 2005.

[Saad 03] Y. Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, 2nd Ed., April 2003.

[Safr 06] I. Safro, D. Ron, and A. Brandt. "Graph minimum linear arrangement by multilevel weighted edge contractions". *J. Algorithms*, Vol. 60, No. 1, pp. 24–41, 2006.

[Saga 94] H. Sagan. *Space Filling Curves*. Springer-Verlag, 1994.

[Sand 07] P. Sanders, K. Mehlhorn, R. Möhring, B. Monien, P. Mutzel, and D. Wagner. "Algorithm Engineering – An Attempt at a Definition". In: *Proceedings of the Oberwolfach Workshop Algorithm Engineering, Math. Forschungsinstitut Oberwolfach Report No. 25/2007*, pp. 1386–1387, 2007.

[Scha 03] S. Schamberger. "Improvements to the Helpful-Set Heuristic and a New Evaluation Scheme for Graph-Partitioners". In: *International Conference on Computational Science and its Applications (ICCSA'03)*, pp. 49–59, Springer-Verlag, 2003.

[Scha 04a] S. Schamberger. "On Partitioning FEM Graphs using Diffusion". In: *Proceedings of the HPGC Workshop of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04)*, IEEE Computer Society, 2004.

[Scha 04b] S. Schamberger and J.-M. Wierum. "A Locality Preserving Graph Ordering Approach for Implicit Partitioning: Graph-Filling Curves". In: *Proceedings of the 17th International Conference on Parallel and Distributed Computing Systems (PDCS'04)*, pp. 51–57, ISCA, 2004.

[Scha 05] S. Schamberger. "A Shape Optimizing Load Distribution Heuristic for Parallel Adaptive FEM Computations". In: *8th International Conference on Parallel Computing Technologies (PaCT'05)*, pp. 263–277, Springer-Verlag, 2005.

[Scha 06] S. Schamberger. *Shape Optimized Graph Partitioning*. PhD thesis, Universität Paderborn, 2006.

[Schl 00] K. Schloegel, G. Karypis, and V. Kumar. "A unified algorithm for load-balancing adaptive scientific simulations". In: *Proceedings of Supercomputing 2000*, p. 59 (CD), IEEE Computer Society, 2000.

[Schl 01] K. Schloegel, G. Karypis, and V. Kumar. "Wavefront Diffusion and LMSR: Algorithms for Dynamic Repartitioning of Adaptive Meshes". *IEEE Transactions on Parallel and Distributed Systems*, Vol. 12, No. 5, pp. 451–466, 2001.

[Schl 02] K. Schloegel, G. Karypis, and V. Kumar. "Parallel static and dynamic multi-constraint graph partitioning". *Concurrency and Computation: Practice and Experience*, Vol. 14, No. 3, pp. 219–240, 2002.

[Schl 03] K. Schloegel, G. Karypis, and V. Kumar. "Graph Partitioning for High Performance Scientific Simulations". In: *The Sourcebook of Parallel Computing*, pp. 491–541, Morgan Kaufmann, 2003.

[Schl 97] K. Schloegel, G. Karypis, and V. Kumar. "Multilevel diffusion schemes for repartitioning of adaptive meshes". *J. Parallel Distrib. Comput.*, Vol. 47, No. 2, pp. 109–124, 1997.

[Seli 84] S. Z. Selim and M. A. Ismail. "K-means-type algorithms: a generalized convergence theorem and characterization of local optimality". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 6, pp. 81–87, 1984.

[Shew 94] J. R. Shewchuk. "An Introduction to the Conjugate Gradient Method Without the Agonizing Pain". Tech. Rep. CMU-CS-94-125, School of Computer Science, Carnegie Mellon University, 1994.

[Shi 00] J. Shi and J. Malik. "Normalized Cuts and Image Segmentation". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 22, No. 8, pp. 888–905, 2000.

[Sima 06] J. Šíma and S. E. Schaeffer. "On the NP-completeness of some graph cluster measures". In: *Proceedings of the 32nd International Conference on Current Trends in Theory and Practice of Informatics (SOFSEM'06)*, pp. 530–537, Springer-Verlag, 2006.

[Simo 97] H. D. Simon and S.-H. Teng. "How Good is Recursive Bisection?". *SIAM J. Sci. Comput.*, Vol. 18, No. 5, pp. 1436–1445, 1997.

[Sope 04] A. J. Soper, C. Walshaw, and M. Cross. "A Combined Evolutionary Search and Multilevel Optimisation Approach to Graph Partitioning". *J. Global Optimization*, Vol. 29, No. 2, pp. 225–241, 2004.

[Ster 06] H. D. Sterck, U. M. Yang, and J. J. Heys. "Reducing Complexity in Parallel Algebraic Multigrid Preconditioners". *SIAM J. Matrix Anal. Appl.*, Vol. 27, No. 4, pp. 1019–1039, 2006.

[Stie 86] T. J. Stieltjes. "Sur les racines de l'équation $X_n = 0$". *Acta Math.*, Vol. 9, pp. 385–400, 1886.

[Stub 00] K. Stüben. "An introduction to algebraic multigrid". In: U. Trottenberg, C. W. Oosterlee, and A. Schüller, Eds., *Multigrid*, pp. 413–532, Academic Press, 2000. Appendix A.

[Stub 01] K. Stüben. "A review of algebraic multigrid". *J. Comput. Appl. Math.*, Vol. 128, No. 1-2, pp. 281–309, 2001.

[The 02] The BlueGene/L Team. "An Overview of the BlueGene/L Supercomputer". In: *Proceedings of the 2002 ACM/IEEE Conference on Supercomputing*, pp. 1–22, ACM, 2002.

[Tref 97] L. N. Trefethen and D. Bau. *Numerical Linear Algebra*. SIAM, June 1997.

[Trot 00] U. Trottenberg, C. W. Oosterlee, and A. Schüller. *Multigrid*. Academic Press, 2000.

[Vand 95] D. Vanderstraeten, R. Keunings, and C. Farhat. "Beyond Conventional Mesh Partitioning Algorithms and the Minimum Edge Cut Criterion: Impact on Realistic Applications". In: *Proceedings of the Seventh SIAM Conference on Parallel Processing for Scientific Computing (PPSC'95)*, pp. 611–614, SIAM, 1995.

[Virn 07] E. Virnik. "An Algebraic Multigrid Preconditioner for a Class of Singular M-Matrices". *SIAM J. Sci. Comput.*, Vol. 29, No. 5, pp. 1982–1991, 2007.

[Wals 00] C. Walshaw and M. Cross. "Parallel Optimisation Algorithms for Multilevel Mesh Partitioning". *J. Parallel Computing*, Vol. 26, No. 12, pp. 1635–1660, 2000.

[Wals 07a] C. Walshaw and M. Cross. "JOSTLE: Parallel Multilevel Graph-Partitioning Software – An Overview". In: F. Magoules, Ed., *Mesh Partitioning Techniques and Domain Decomposition Techniques*, pp. 27–58, Civil-Comp Ltd., 2007. (Invited chapter).

[Wals 07b] C. Walshaw. "The Graph Partitioning Archive". `http://staffweb.cms.gre.ac.uk/~c.walshaw/partition/`, 2007.

[Wals 95] C. Walshaw, M. Cross, and M. G. Everett. "A Localised Algorithm for Optimising Unstructured Mesh Partitions". *International J. Supercomputer Appl.*, Vol. 9, No. 4, pp. 280–295, 1995.

[Wals 97] C. Walshaw, M. Cross, and M. G. Everett. "Parallel Dynamic Graph Partitioning for Adaptive Unstructured Meshes". *J. Parallel Distributed Computing*, Vol. 47, No. 2, pp. 102–108, 1997.

[Xu 97] C. Xu and F. C. M. Lau. *Load Balancing in Parallel Computers.* Kluwer, 1997.

[Yen 05] L. Yen, D. Vanvyve, F. Wouters, F. Fouss, M. Verleysen, and M. Saerens. "Clustering using a random-walk based distance measure". In: *Proceedings of the 13th European Symposium on Artificial Neural Networks (ESANN'05)*, pp. 317–324, 2005.

[Zach 77] W. W. Zachary. "An information flow model for conflict and fission in small groups". *Journal of Anthropological Research*, Vol. 33, pp. 452–473, 1977.

[Zha 01] H. Zha, X. He, C. H. Q. Ding, M. Gu, and H. D. Simon. "Spectral Relaxation for K-means Clustering". In: *Proceedings of Advances in Neural Information Processing Systems 14 (NIPS'01)*, pp. 1057–1064, MIT Press, 2001.

[Zhao 03] Y. Zhao and G. Karypis. "Clustering in the life sciences". In: M. Brownstein, A. Khodursky, and D. Conniffe, Eds., *Functional Genomics: Methods and Protocols*, pp. 183–218, Humana Press, 2003.

[Zumb 03] G. Zumbusch. *Parallel Multilevel Methods: Adaptive Mesh Refinement and Loadbalancing.* Teubner, 2003.

A. Appendix

A.1. Bubble-FOS/C: Additional Experimental Results

Table A.1.: Comparison of Bubble-FOS/C in different parameter settings for ℓ_1-norm.

	AC2/CO1		AC2/CO2		AC3/CO2		AC3/CO3	
k	EC	bnd	EC	bnd	EC	bnd	EC	bnd
4	931.18	995.09	885.29	960.81	876.53	956.94	**868.84**	**942.95**
8	1406.66	1527.39	1381.44	1499.61	1374.35	1492.91	**1365.20**	**1483.63**
12	1855.50	1998.73	1841.18	1978.25	1837.19	1970.00	**1826.61**	**1957.74**
16	2191.84	2391.20	2164.10	2356.75	2155.86	2346.54	**2148.34**	**2334.14**
20	2535.04	2755.33	2494.36	2718.29	2486.95	2714.53	**2472.36**	**2699.26**
avg	1784.04	1933.55	1753.27	1902.74	1746.18	1896.18	**1736.27**	**1883.54**

Table A.2.: Comparison of Bubble-FOS/C in different parameter settings for ℓ_∞-norm

	AC2/CO1		AC2/CO2		AC3/CO2		AC3/CO3	
k	ext	bnd	ext	bnd	ext	bnd	ext	bnd
4	548.89	293.98	518.23	279.85	523.69	281.79	**517.63**	**278.80**
8	438.24	237.64	431.66	234.06	427.11	**231.28**	**426.24**	232.10
12	401.58	215.44	393.93	209.45	**389.70**	208.48	393.15	**206.75**
16	345.94	185.73	343.66	183.41	338.86	181.05	**338.70**	**179.41**
20	328.53	175.63	325.19	175.48	**321.26**	**173.65**	324.48	174.86
avg	412.63	221.68	402.53	216.45	400.13	215.25	**400.04**	**214.39**

Table A.3.: Comparison of Bubble-FOS/C using AMG with Bubble-FOS/C using CG for ℓ_1- and ℓ_∞-norm and AC3/CO2.

	Bubble-FOS/C with CG and AC3/CO2				Bubble-FOS/C with AMG and AC3/CO2			
k	EC	bnd_1	ext_∞	bnd_∞	EC	bnd_1	ext_∞	bnd_∞
4	**870.3**	**954.5**	529.2	282.2	876.5	956.9	**523.7**	**281.8**
8	**1368.7**	**1480.6**	**426.1**	**230.5**	1374.4	1492.9	427.1	231.3
12	**1833.3**	**1968.2**	392.9	**207.4**	1837.2	1970.0	**389.7**	208.5
16	**2150.0**	**2336.1**	**336.4**	**180.5**	2155.9	2346.5	338.9	181.1
20	**2475.4**	**2701.1**	322.0	174.3	2487.0	2714.5	**321.3**	**173.7**
avg	**1739.5**	**1888.1**	401.3	**215.0**	1746.2	1896.2	**400.1**	215.3

Table A.4.: Comparison of Bubble-FOS/C using AMG without and with the virtual vertex ($\phi = 1/512$) for ℓ_1- and ℓ_∞-norm and AC3/CO3.

	Bubble-FOS/C (AMG)				Bubble-FOS/C (AMG) with virtual vertex			
k	EC	bnd_1	ext_∞	bnd_∞	EC	bnd_1	ext_∞	bnd_∞
4	868.8	943.0	**517.6**	278.8	**868.7**	**934.2**	526.6	**278.5**
8	1365.2	1483.6	**426.2**	232.1	**1361.0**	**1468.9**	429.2	**231.0**
12	1826.6	1957.7	**393.2**	**206.8**	**1818.0**	**1954.9**	399.1	213.3
16	2148.3	2334.1	338.7	**179.4**	**2134.8**	**2322.4**	**337.1**	180.7
20	**2472.4**	2699.3	324.5	174.9	2474.1	**2678.6**	**321.7**	**172.2**
avg	1736.3	1883.5	**400.0**	**214.4**	**1731.3**	**1871.8**	402.7	215.1

A.2. DibaP: Additional Experimental Results

Table A.5.: Average edge-cut, boundary nodes, and migration volume in the ℓ_1-norm for repartitionings computed by ParMETIS, Jostle, and DibaP on twelve small graph sequences.

	ParMETIS		Jostle		DibaP	
Sequence	EC	bnd_1	EC	bnd_1	EC	bnd_1
bubbles	366.7	723.8	323.8	638.6	**312.8**	**612.1**
change	357.9	706.5	308.4	607.6	**297.9**	**588.8**
circles	371.2	733.0	328.8	646.4	**314.2**	**610.0**
fastrot	433.6	857.4	385.2	757.9	**362.2**	**705.9**
fasttric	455.0	900.1	407.5	803.2	**376.3**	**741.8**
heat	182.2	360.2	**154.5**	**304.2**	159.5	306.4
refine	225.9	448.6	199.9	389.1	**191.3**	**377.6**
ring	274.4	541.1	238.0	471.2	**231.0**	**446.5**
rotation	387.9	767.7	342.6	675.3	**341.0**	**662.7**
slowrot	431.7	853.5	383.5	754.9	**359.3**	**703.9**
slowtric	502.1	994.0	434.2	856.5	**406.7**	**796.5**
trace	328.1	644.1	285.4	557.9	**273.6**	**524.8**

Table A.6.: Average number of external edges, boundary nodes, and migration volume in the ℓ_∞-norm for repartitionings computed by ParMETIS, Jostle, and DibaP on twelve small graph sequences.

	ParMETIS		Jostle		DibaP	
Sequence	ext_∞	bnd_∞	ext_∞	bnd_∞	ext_∞	bnd_∞
bubbles	86.0	84.8	75.4	74.1	**66.0**	**64.3**
change	83.0	81.5	70.9	69.6	**64.6**	**63.6**
circles	81.1	80.1	74.4	72.9	**66.7**	**63.7**
fastrot	96.1	94.8	86.4	84.7	**73.6**	**71.4**
fasttric	98.4	97.1	92.1	90.4	**77.6**	**76.3**
heat	56.9	56.2	**45.1**	**44.4**	48.1	46.6
refine	47.6	46.6	42.9	41.2	**36.2**	**35.7**
ring	71.3	69.9	61.1	60.5	**59.9**	**57.1**
rotation	90.0	88.7	80.3	78.8	**71.3**	**70.0**
slowrot	93.2	91.9	82.2	80.7	**70.4**	**68.7**
slowtric	112.9	111.6	97.6	95.8	**78.5**	**76.4**
trace	73.8	71.9	65.9	63.8	**58.9**	**56.3**

Table A.7.: Average number of external edges and boundary nodes in the ℓ_1- and ℓ_∞-norm for repartitionings computed by ParMETIS, Jostle, and DibaP on three large graph sequences.

	ParMETIS		Jostle		DibaP	
Sequence / norm	ext	bnd	ext	bnd	ext	bnd
bigtric (ℓ_1)	1866.3	3717.7	1569.4	3121.9	**1436.8**	**2865.9**
bigtric (ℓ_∞)	321.3	319.6	267.0	265.5	**230.9**	**229.7**
bigbubbles (ℓ_1)	4716.2	9387.2	3974.8	7873.7	**3527.1**	**7041.7**
bigbubbles (ℓ_∞)	845.7	840.3	740.4	729.0	**615.4**	**613.5**
bigtrace (ℓ_1)	4124.9	8212.2	3349.1	6630.9	**2919.7**	**5815.5**
bigtrace (ℓ_∞)	718.7	713.2	584.5	577.8	**463.9**	**461.7**

A.3. Description of Random Graphs with Planted Partitions (Model 2)

Table A.8.: Parameters (size n, intra-cluster edge probability p_{int}, inter-cluster edge probability p_{ext}) and resulting node degree values for randomly generated graphs with planted partitions and $k = 9$. Note: $p_{int} \in \{\frac{1200}{n \cdot k}, \frac{2500}{n \cdot k}\}$, $p_{ext} \in \{\frac{110}{n \cdot k}, \frac{150}{n \cdot k}\}$.

$k = 9$, Setting	1	2	3	4	5	6	7	8	9
$n \cdot 10^4$	2	2	2	4	4	4	8	8	8
$p_{int} \cdot 10^{-3}$	6.667	13.889	13.889	3.333	6.944	6.944	1.667	3.472	3.472
$p_{ext} \cdot 10^{-3}$	0.611	0.611	0.833	0.306	0.306	0.417	0.153	0.153	0.208
minimum degree	6	9	13	3	7	10	5	6	11
maximum degree	56	83	88	57	92	95	62	89	95
average degree	28.396	47.642	51.467	28.427	47.698	51.555	28.398	47.696	51.545

Table A.9.: Parameters (size n, intra-cluster edge probability p_{int}, inter-cluster edge probability p_{ext}) and resulting node degree values for randomly generated graphs with planted partitions and $k = 13$. Note: $p_{int} \in \{\frac{1200}{n \cdot k}, \frac{2500}{n \cdot k}\}$, $p_{ext} \in \{\frac{110}{n \cdot k}, \frac{150}{n \cdot k}\}$.

$k = 13$, Setting	1	2	3	4	5	6	7	8	9
$n \cdot 10^4$	2	2	2	4	4	4	8	8	8
$p_{int} \cdot 10^{-3}$	4.615	9.615	9.615	2.308	4.808	4.808	1.154	2.404	2.404
$p_{ext} \cdot 10^{-3}$	0.423	0.423	0.577	0.212	0.212	0.288	0.106	0.106	0.144
minimum degree	1	3	5	2	3	4	1	3	4
maximum degree	35	52	55	36	51	57	37	53	57
average degree	15.972	24.893	27.691	16.034	24.987	27.783	15.972	24.948	27.761

Table A.10.: Parameters (size n, intra-cluster edge probability p_{int}, inter-cluster edge probability p_{ext}) and resulting node degree values for randomly generated graphs with planted partitions and $k = 17$. Note: $p_{int} \in \{\frac{1200}{n \cdot k}, \frac{2500}{n \cdot k}\}$, $p_{ext} \in \{\frac{110}{n \cdot k}, \frac{150}{n \cdot k}\}$.

$k = 17$, Setting	1	2	3	4	5	6	7	8	9
$n \cdot 10^4$	2	2	2	4	4	4	8	8	8
$p_{int} \cdot 10^{-3}$	3.529	7.353	7.353	1.765	3.676	3.676	0.882	1.838	1.838
$p_{ext} \cdot 10^{-3}$	0.324	0.324	0.441	0.162	0.162	0.221	0.081	0.081	0.110
minimum degree	1	2	3	1	2	2	1	1	1
maximum degree	29	37	40	28	39	41	30	39	42
average degree	10.796	15.981	18.201	10.836	15.989	18.188	10.781	15.957	18.144

A.4. DibaP: Best-known Edge-cut Results

Table A.11.: Instances of benchmark graphs for which DibaP has computed the currently best-known edge-cut values (marked by X, 84 in total out of 144 possible ones for these six graphs) for the specified numbers of partitions k in different imbalance settings. Last update: 29 Feb 2008.

Imbalance	0%					1%					3%					5%				
Graph / k	4	8	16	32	64	4	8	16	32	64	4	8	16	32	64	4	8	16	32	64
tooth		X	X	X			X		X			X					X			
598a	X	X	X	X		X		X	X	X				X						
144		X	X	X	X			X	X					X	X				X	X
wave		X	X	X	X		X	X	X	X	X	X	X	X	X	X	X	X	X	X
m14b	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
auto	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X