Dissertation

Local Strategies for Swarm Formations on a Grid

Daniel Jung

Paderborn University

Reviewers:

- Prof. Dr. Friedhelm Meyer auf der Heide,
  Paderborn University
- Prof. Dr. Christian Scheideler,
  Paderborn University

Für meine Eltern.

Contents

Vorwort

Ich bedanke mich bei meinem Doktorvater Friedhelm Meyer auf der Heide für seine stets motivierende Betreuung und dafür, dass er mir die Möglichkeit zur Promotion gab. Sämtlichen Kolleginnen und Kollegen danke ich für eine angenehme und erfolgreiche Zusammenarbeit. Dies gilt insbesondere allen meinen Koautorinnen und Koautoren. Matthias Fischer danke ich darüber hinaus für die gelungene, gemeinsame Arbeit bei unseren Lehretätigkeiten. Diese hat mir stets große Freude bereitet.

Dank gilt auch Rolf Klein, Elmar Langetepe und allen übrigen Mitarbeiterinnen und Mitarbeitern deren Arbeitsgruppe an der Universität Bonn, da sie mein Interesse am Forschungsbereich meiner Promotion geweckt haben.

Privat danke ich allen meinen Freunden, die mir ein ausgleichender Gegenpol zur Arbeit waren. Ganz besonders danke ich jedoch meinen Eltern für ihre stets verständnis- und aufopferungsvolle Unterstützung auf meinem Weg durch Schule, Studium und Promotion. Ohne sie wäre all dies nicht möglich gewesen. Dank gilt auch meinem Bruder Michael, der mich zur Aufnahme eines Informatikstudiums bewogen hat. Nicht zuletzt danke ich meiner Freundin Katja, die im Promotionsstress stets eine wichtige, verständnisvolle Partnerin war.

Daniel Jung
Paderborn, Dezember 2017

Zusammenfassung

Meine Dissertation beschäftigt sich mit dem Gathering Problem für Schwärme von n punktförmigen Robotern auf einem Gitter, bei dem sich alle Roboter des Schwarms auf einem zuvor nicht festgelegten Punkt versammeln sollen. Besonderes Augenmerk liegt auf der starken Einschränkung der Roboterfähigkeiten. Hierzu zählen insbesondere das Fehlen einer globalen Steuerung, eines globalen Kompasses, einer globalen Sichtweite und einer (globalen) Kommunikationsfähigkeit. Darüber hinaus sind alle Roboter identisch. Den Robotern sind nur lokale Fähigkeiten gegeben. Hierzu zählt etwa eine nur konstante Sichtweite. Die Roboter arbeiten alle vollständig synchron.

In der Arbeit präsentieren und analysieren wir drei unterschiedliche Gatheringstrategien unter verschiedenen Robotermodellen. Wir beweisen jeweils formal Korrektheit und Gesamtlaufzeit: In Kapitel 4 liegt der Fokus auf der Minimierung der zur Verfügung stehenden Roboterfähigkeiten. Die zugrundeliegende Strategie schließt das Gathering in Zeit $\mathcal{O}(n^2)$ ab. In den darauffolgenden Kapiteln 5 und 6 ist das Ziel die Optimierung der Gesamtlaufzeit unter weiterhin ausschließlich lokalen Roboterfähigkeiten: Wir erlauben zusätzlich einen konstant großen Speicher und eine konstante Anzahl lokal sichtbarer Status (Lichter, Flaggen). Für die Strategien beider Kapitel zeigen wir eine asymptotisch optimale Laufzeit von $\mathcal{O}(n)$. Während in Kapitel 4 und 5 Sicht und Schwarmzusammenhang intuitiv über Nachbarschaften auf dem Gitter definiert sind, unterscheidet sich dies in Kapitel 6: Hier sind die Roboter von Beginn an zu einer Kette verbunden, bei der je zwei Kettennachbarn Abstand 1

voneinander haben müssen. Ein Roboter kann dabei nur eine konstante Anzahl seiner direkten Kettennachbarn sehen und auch nur mit diesen interagieren. Aber auch unter diesem Modell ist Gathering in $\mathcal{O}(n)$ möglich.

Abstract

My dissertation deals with the Gathering problem for swarms of n point-shaped robots on a grid, in which all robots of the swarm are supposed to gather at a previously undefined point. Special attention is paid to the strong limitation of robot capabilities. These include in particular the lack of global control, a global compass, global visibility and (global) communication skills. Furthermore, all robots are identical. The robots are given only local abilities. This includes a constant range of vision. The robots all work completely synchronously.

In this work we present and analyze three different Gathering strategies in different robot models. We formally prove correctness and total running time: Chapter 4 focuses on minimizing the available robot capabilities. The underlying strategy completes the gathering in $\mathcal{O}(n^2)$ time. For the following Chapters 5 and 6, the aim is to optimize the total running time under using only local robot capabilities: We additionally allow a constant-sized memory and a constant number of locally visible statuses (lights, flags). For the strategies of both chapters we show an asymptotically optimal running time of $\mathcal{O}(n)$. While in Chapters 4 and 5, vision and swarm connectivity are intuitively defined by neighborhoods on the grid, this differs in Chapter 6: Here, the robots are connected to a chain from the beginning, where two chain neighbors must have a distance of 1 from each other. A robot can only see and interact with a constant number of its direct chain neighbors. But also in this model, Gathering is possible in $\mathcal{O}(n)$.

CHAPTER 1

Introduction

Imagine a space mission where one or more robots have to perform some tasks on an outer planet. For example, missions to the Mars planet are a popular topic. Here, the *Opportunity* and *Curiosity* rovers are good examples of successful inspecting mobile robots. They carry lots of sensors, tools and computational power and can deliver many useful data. But landing such a big and sensitive robot without crashing is an extremely difficult task. Furthermore, over these large distances, the slow transfer rate of remote controlling signals does not allow an even approximate real-time remote control. The same applies to the remote controlled acting after the landing. These tasks require the collective work of many high qualified researchers, engineers and technicians.

Much less effort would be required if robots would be simpler, smaller, lighter, and perform their tasks collectively as intelligent swarms and autonomously, i.e., without the requirement of continuous remote control. Lighter robots further require less fuel during transportation to the outer planet. If robots are simpler, then clearly the manufacturing costs are reduced. This can be continued if all robots are equal and furthermore do not even need to be individually programmed.

Because no infrastructure is present on an outer planet, the robots can rely

only on the local information that they can obtain by themselves.

A different, more futuristic scenario is medical technology: In the meantime, the digestive system is sometimes examined using a small robot. If we farsightedly assume to examine blood vessels, then using only one robot would take too much time to explore and robots must be even smaller. Swarms of tiny robots are required, but precisely controlling them from outside the human's body is difficult. Also here autonomous acting is desired. While doing this, robots have to rely on very restricted information about their environment.

In all cases, for being able to reasonably apply robot strategies also to bigger swarms, the robot strategies must be reasonably scalable concerning the working speed.

Research on robot swarms commonly deals with formation tasks. Christensen et al. [COD07] present a distributed algorithm with which a swarm of mobile real robots—so-called "s-bots" [Mon+05]—can form big swarm formation shapes, with a vision only up to a constant distance and without communication. These robots are equipped with a gripper that allows them, besides grabbing other objects, to cling to a neighboring robot.

In our community, robot swarms are considered from a higher abstraction level. Robots are usually point-shaped or represented as circles. An important question is the minimum requirements of robot capabilities (vision, communication, memory, …) for solving basic formation tasks like circle and line formation or the very basic gathering formation task, where all robots have to gather at some previously undefined point. Feasibility is formally proven. When minimizing robot capabilities, a common procedure is to restrict the global information (vision, positioning, compass, communication) that is available to the robots.

Another important topic is the formal proofs of upper and lower bounds for the total running time and moved distances that are needed for performing the considered formation task.

A comprehensive collection of basic robot models and state-of-the-art strategies, concentrating on formal analysis, can be found in the book of Flocchini, Prencipe and Santoro [FPS12].

The focus of my research was the development of local swarm formation strategies for point-shaped, anonymous, autonomous mobile robots, including the formal proofs of their correctness and running times. I mainly considered

the most basic formation problem: the gathering. Under this topic, research is mainly done for robots in the Euclidean plane. Unlike this, I always assume that robots are located on a two-dimensional grid. This means that my strategies are location- as well as time-discrete. On the one hand, using basic robot models, I concentrated on the development of local strategies that optimize the total running times. On the other hand, I focussed on the development of local strategies that still work if the robot capabilities are even more restricted concerning these local capabilities, while analyzing the impact on the total running times.

Concerning robot capabilities, "local" means that a robot only has local information about its environment (and can also execute only local operations). The most important of these local properties are that a robot does not have a global compass (and has no global sense of chirality) and can see other robots only within a constant distance, which is often called the robot's *viewing radius*. The part of the environment (restricted by the viewing radius) that is visible to a robot is called the robot's *viewing range*.

For local strategies, the connectivity of the swarm is an important property. If connectivity is given by vision, then the swarm's connectivity would break if it splits up into two subswarms such that no robot of one subswarm can see any robot of the other subswarm. Then, these two subswarms could never "find" each other for reassembling to only one swarm and the desired formation cannot be formed. That is why our strategies ensure that connectivity does not break.

Robot operations are commonly split into three steps [CP04]: *look*, *compute*, *move*. In the *look* step, the robot gets a snapshot of its viewing range, i.e., the current scenario from its own perspective, restricted by its constant viewing radius. During the *compute* step, the robot computes its action, and eventually performs it in the *move* step. We say it executes Look-Compute-Move or LCM cycles. The time model defines how these steps of different robots are aligned. In this thesis, we use the fully synchronous $\mathcal{FSYNC}$ time model, where all robots execute their LCM cycles simultaneously.

1.1 Results of the Thesis

We have developed and analyzed three different local gathering strategies for point-shaped robots on a two-dimensional infinite grid.

Gathering Anonymous, Oblivious Robots on a Grid

In Chapter 4, we use the same time and *Basic&Plain* robot model as in the so far fastest Euclidean plane strategy Go-To-The-Center published by Degener et al. [Deg+11] who use the algorithmic idea from Ando et al. [ASY95]. My strategy instead works on the grid.

The robot actions of our strategy are rather simple. Instead the correctness and running time proofs are very complicated. For this, a combination of three different progress measures is required.

We prove that the worst-case running time of our Oblivious-Gather grid strategy matches the $\mathcal{O}(n^2)$ bound of Go-To-The-Center under the same time model and robot model.

The contents of this chapter have been published in the following paper:

> D. Jung, M. Fischer and F. Meyer auf der Heide. "Gathering Anonymous, Oblivious Robots on a Grid". In: *Algorithms and Experiments for Wireless Sensor Networks – 13th International Symposium, ALGOSENSORS 2017, Vienna, Austria, September 7-8, 2017. Proceedings*, cf. [FJM17a]

The full version of the paper can be found at:

> D. Jung, M. Fischer and F. Meyer auf der Heide. "Gathering Anonymous, Oblivious Robots on a Grid". In: *CoRR abs/1702.03400 (2017)*, cf. [FJM17b]

Asymptotically Optimal Gathering on a Grid

In Chapter 5, we significantly improve the running time by a factor of n to $\mathcal{O}(n)$. For this, we allow two additional robot capabilities: a constant-sized memory and a constant number of states that are locally visible to other robots in viewing range (*Extended Basic&Plain* robot model).

The presence of a (constant-sized) memory allows a robot to include its actions from a constant number of previous LCM cycles when calculating its next action. This enables much more complex and effective robot operations. The memory allows robot tasks that take sequences of many sequential robot operations until they finally enable a gathering progress. This significantly complicates the strategy. As a reparation for this, the formal correctness and running time proofs become more straightforward. We show that much of the work can be parallelized so that the Grid-Gather strategy significantly improves the total running time needed for the gathering by a factor of n to $\mathcal{O}(n)$, which is also asymptotically optimal for the problem in general. The contents of this chapter have been published in the following paper:

> D. Jung, A. Cord-Landwehr, M. Fischer and F. Meyer auf der Heide. "Asymptotically Optimal Gathering on a Grid". In: *Parallelism in Algorithms and Architectures – 28th ACM Symposium, SPAA 2016, Pacific Grove, California, USA, July 11-13, 2016. Proceedings*, cf. [Cor+16]

Gathering a Closed Chain of Robots on a Grid

In Chapter 6, we also use the additional capabilities constant-sized memory and constant number of locally visible states for a different, more restricted connectivity and visibility definition (*Extended Basic&Plain Closed-Chain* robot model): Robots are always connected as an initially given, possibly self-intersecting closed chain and also vision is restricted to chain neighborhoods.

This introduces the difficulty that robots which are located at the same or close together located positions in general have completely different viewing ranges and so also behave completely differently.

For the Closed-Chain-Gather strategy, we prove the asymptotically optimal running time $\mathcal{O}(n)$. The contents of this chapter have been published in the following paper:

> D. Jung, S. Abshoff, A. Cord-Landwehr, M. Fischer and F. Meyer auf der Heide. "Gathering a Closed Chain of Robots on a Grid". In: *International Parallel and Distributed Processing – 30th IEEE International Symposium, IPDPS 2016, Chicago, Illinois, USA, May 23-27, 2016. Proceedings*, cf. [Abs+16]

1.2 Animations

In order to get a better impression of the gathering strategies presented in this thesis, we provide some video animations and, on the right page margin, a flip book.

The simulators of the Closed-Chain-Gather and Grid-Gather strategy have been developed by André Graute and Marcel Stienemeier, whose bachelor theses I supervised [Gra16; Sti16]. André Graute developed the simulator for the Closed-Chain-Gather strategy. The Grid-Gather simulator was developed by Marcel Stienemeier.

1.2.1 Flip book

The flip book animations, in order from top to bottom, show examples of the Oblivious-Gather, Grid-Gather and Closed-Chain-Gather strategy.

Oblivious-Gather strategy. In this example, the Oblivious-Gather strategy is applied to some valid random swarm. The red marked robots perform a hop. The green cells mark the target cells of the corresponding hops.

Grid-Gather strategy. In this example, though the Grid-Gather strategy also works on arbitrary connected swarms, we chose an unfilled rectangle as an example, because this shape gives a very good impression of behavior that significantly increases the gathering speed. One can see that multiple moving visible states that initiate hops are active in parallel and can also be nested.

Red arrows denote the moving directions of the visible states. Green arrows point to the target cells of the initiated hops.

Concerning this strategy, one cannot get an impression of the functioning in small examples, because the formal proofs require large constants. Therefore, in the given flip book example, we reduced some of these constants. Animations of this strategy that use the correct constants are provided as online videos (Section 1.2.2).

Closed-Chain-Gather strategy. This example shows a simulation of the Closed-Chain-Gather strategy, applied to a valid random swarm. Grey dots mark cells that contain more than one robot (intersections, overlappings).

Orange colored dots mark visible state positions with an arrow head pointing in their moving directions. Green dots mark target cells of certain operations.

1.2.2 Video animations

The video animations can be found at YouTube (playlist):

https://www.youtube.com/playlist?list=PLsqmqCMaaewCbdLLiDEhxNGlSkIhXTiHA

and http://roboterschubser.de

Every strategy is applied to a random and to a 200×200 square-shaped swarm. We point out notable behavior:

Oblivious-Gather strategy. In the 200×200 square example, the diagonal edges seem to be flickering. Taking a closer look, one can notice that the cause is that lots of robots hop back towards the swarm's outside.

Grid-Gather strategy. In contrast to the corresponding Oblivious-Gather animation, the 200×200 square is gathered in a very targeted way.

For sake of illustration, in the random example the swarm's outer boundary is marked in black color.

Closed-Chain-Gather strategy. In the 200×200 square example the behavior is quite similar to the corresponding Grid-Gather example. The reason is that in this example the connectivity and vision are similar for both strategies.

CHAPTER 2

Time Model and Robot Model

Concerning swarm formation problems, the available robot capability types (robot model) and the alignment of the robots' operations (time model) in general have a high impact on parameters like solvability, running time and the distance that must be traveled by a robot.

In this chapter, we explain in detail the time model and the robot models that are used in this thesis. We start with the time model.

2.1 Fully Synchronous Time Model

In the whole thesis, we consider the fully synchronous $\mathcal{FSYNC}$ time model. Here, time is subdivided into equally sized rounds of constant lengths. All robots are always active and perform all operations synchronously. In every round all robots simultaneously execute their operations in the common *look-compute-move model* [CP04], which divides one operation into three steps: In the *look* step, the robot gets a snapshot of the current scenario from its own perspective, restricted to its constant-sized viewing range. During the *compute* step, the robot computes its action, and eventually performs it in the *move* step. We say it executes Look-Compute-Move or LCM cycles.

In contrast to asynchronous time models, the synchrony of the $\mathcal{FSYNC}$ model allows meaningful running time estimations by counting these LCM cycles. But synchrony also introduces many symmetry issues that, for example, can break the swarm's connectivity or lead to unwanted robot operations. This significantly complicates strategy design.

2.2 Local Robot Model

My research only deals with so-called *local* strategies. In the following we will substantiate the meaning of *local* in this thesis.

2.2.1 Common robot capabilities used in this thesis

I use a very restricted robot model. I assume that the robots

- do not have a global vision
- do not have a sense of global coordinates
- do not have a global compass
- do not have a global chirality sense
- do not have a global control (autonomous)
- are indistinguishable (anonymous)
- cannot communicate
- are oblivious (only in the Oblivious-Gather strategy (Chapter 4))
- are stateless (only in the Oblivious-Gather strategy (Chapter 4))

Concerning present capabilities, I assume that the robots

- have a local vision: robot vision only up to some fixed constant distance that we call the *viewing radius* for the Oblivious-Gather and Grid-Gather strategy (Chapter 4 resp. 5) and *viewing path length* for the Closed-Chain-Gather strategy (Chapter 6)

- have a constant number of states that are locally visible to other robots (only in the Grid-Gather and Closed-Chain-Gather strategy). Concerning visible states, see Section 2.2.2.
- can execute local operations: i.e.,
  - can move at most a constant distance per step (It can *hop* to one of its eight neighboring grid cells.)
  - can *merge* with other robots (i.e., all but one is removed) if more than one robot is located at the same position
  - can set/unset its own visible states (only in the Grid-Gather and Closed-Chain-Gather strategy)

All these capabilities are local. We call the snapshot of the environment that is visible to a robot, the robot's *viewing range*. Note, that this is bounded by the robot's viewing radius and viewing path length, respectively. Because in this thesis robots are located on a two-dimensional grid, the number of cells respectively robot positions in a robot's viewing range is upper bounded by a fixed constant number.

2.2.2 Visible states

In all my strategies, I forbid communication. This means that a robot cannot even locally communicate to other robots. A robot r can see other robots only if they are located within its viewing range that is upper bounded by a fixed constant distance to r. The only information that r gets about these visible robots are their relative positions w.r.t. its own position and (local) compass, and if included in the corresponding robot model, then also their visible states (flags, lights).

In the Closed-Chain-Gather and the Grid-Gather strategy (Chapter 6 and 5) robots actually have a constant number of states that are locally visible to other robots. This means that, in addition to the relative positions of the robots within its viewing range, a robot can see their visible states. Such visible states are often also called *flags* or *lights*. This robot capability has been suggested by Peleg et al. [Pel05; EP07]. In literature, such robots are often called *luminous*. There is some research done concerning the impact of this luminosity [Das+12; Das+16; Lun+17].

In this thesis, in the Oblivious-Gather strategy (Chapter 4), even visible states are prohibited.

2.2.3 *Basic&Plain* robot model: Oblivious-Gather (Chapter 4)

For the Oblivious-Gather grid strategy, we use the *Basic&Plain* robot model that is used in the comparable Euclidean plane Go-To-The-Center strategy (Degener et al. [Deg+11]). The Oblivious-Gather strategy has the following characteristics, compared to the common robot model, explained in Section 2.2.1. A robot

- does not have visible states
- does not have memory (oblivious)
- has a viewing radius of 7, i.e., can see other robots up to L_1-distance 7

In our strategy two robots are connected if they have L_1-distance 1, i.e., are located at vertically or horizontally neighboring grid cells.

For an example, see Figure 2.1.

If in the underlying time model and robot model multiple robots are located on the same position, then they behave like one robot. For sake of simpler analysis, we say they *merge*.

2.2.4 *Extended Basic&Plain* robot model: Grid-Gather (Chap. 5)

For the Grid-Gather strategy, we extend the *Basic&Plain* robot model that is used in the comparable Euclidean plane Go-To-The-Center strategy (Degener et al. [Deg+11]) and in my Oblivious-Gather strategy. We add a constant number of visible states and a constant memory and call it *Extended Basic&Plain* robot model. The Grid-Gather strategy has the following characteristics, compared to the common robot model, explained in Section 2.2.1. A robot has

- a constant number of visible states
- a constant-sized memory for remembering a constant number of previous actions
- a viewing radius of 19, i.e., can see other robots up to L_1-distance 19

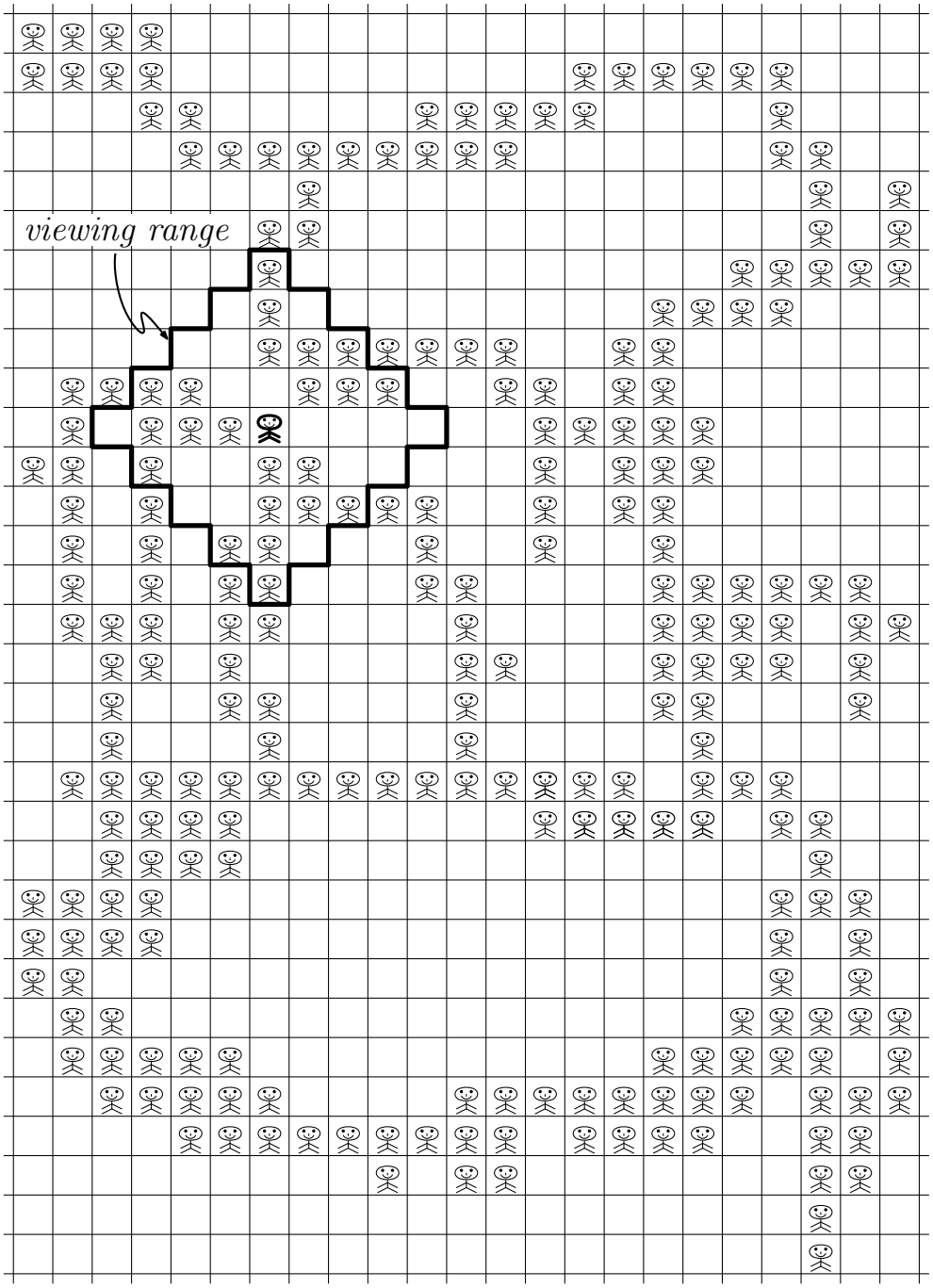


Figure 2.1: Oblivious-Gather: A robot swarm. The fat robot calculates its actions only on the basis of the robot positions in its viewing range (snapshot of the environment). Because there is no global compass and no global chirality, the strategy must work under any rotation and reflection of the snapshot. In this example, for sake of better visualisation, we assume a viewing radius of 4.

In our strategy two robots are connected if they have L_1-distance 1, i.e., are located at vertically or horizontally neighboring grid cells.

Figure 2.2 shows an example for the Grid-Gather strategy, where the robot positions are the same as in the Oblivious-Gather example.

If using the Grid-Gather strategy with the underlying time and robot model multiple robots are located on the same position, then they behave like one robot. For sake of simpler analysis, we say they *merge*.

2.2.5 *Extended Basic&Plain Closed-Chain* robot model: Closed-Chain-Gather (Chapter 6)

The *Extended Basic&Plain Closed-Chain* model that we use for the Closed-Chain-Gather strategy is very similar to the *Extended Basic&Plain* robot model (cf. Section 2.2.4) that we use for the Grid-Gather strategy. The differences between both robot models affect the connectivity and visibility definition: In the *Extended Basic&Plain Closed-Chain* robot model, the robots are always connected as an initially given closed chain, while consecutive chain neighbors must have L_1-distance 1, i.e., be located at vertically or horizontally neighboring grid points. The Closed-Chain-Gather strategy has the following characteristics, compared to the common robot model explained in Section 2.2.1. A robot has

- a constant number of visible states
- a constant-sized memory
- a vision that is restricted to chain neighbors, only. Other robots, even if located at neighboring or the same grid cell, cannot be detected.
- a viewing path length of 11, i.e., can see the next 11 neighbors in both directions along the chain

In the example of Figure 2.3, the same robot and flag positions are chosen as in the example for the Grid-Gather strategy, but the robots are connected by some closed chain (blue). If the chain visits some cell k times, then also k robots are located at this cell. These robots do not see each other if their distance along the chain is bigger than the *viewing path length*. In the example, for sake of better visualization, a robot can see its chain neighbors only up to distance 4 (*viewing path length*) instead of the actual value 11. The fat red line denotes

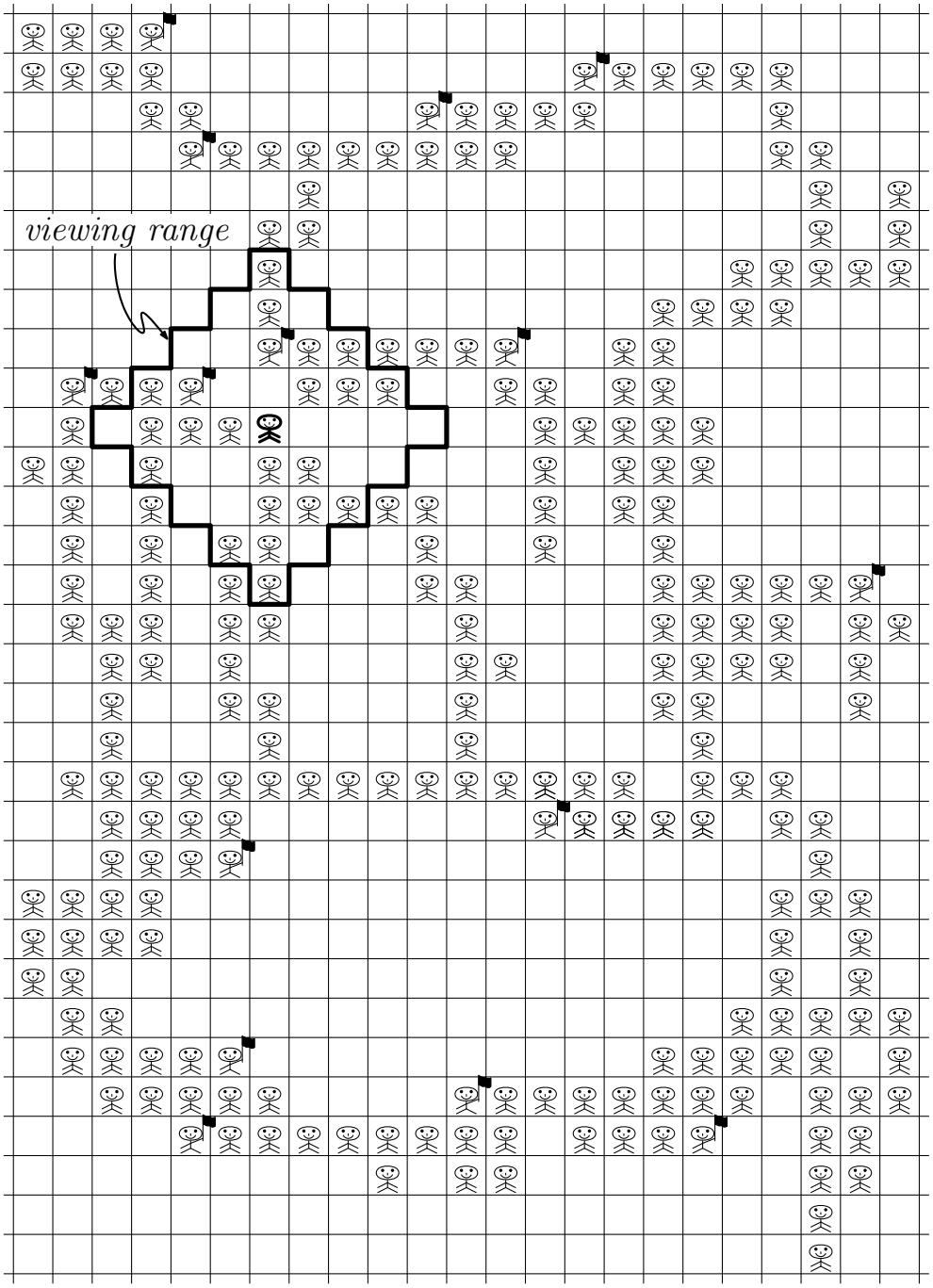


Figure 2.2: GRID-GATHER: A robot swarm. The fat robot calculates its actions on the basis of the robot positions and raised flags in its viewing range (snapshot of the environment). Because there is no global compass and no global chirality, the strategy must work under any rotation and reflection of the snapshot. In this example, for sake of better visualisation, we assume a viewing radius of 4.

the viewing range of the fat robot. The dashed cell contains two robots. One is visible in distance 2, one in distance 3. In contrast to this, on the dotted cell that also contains two robots, only one of them is included in the viewing range, because the distance along the chain of the other one is too big.

Interestingly, in contrast to Oblivious-Gather and Grid-Gather, now robots that occupy the same position see different viewing ranges and as a result also behave differently.

We also want to have a merge operation in the Closed-Chain-Gather strategy. Here, we also let two robots merge if they are located at the same grid point, but only if they are furthermore direct neighbors on the chain. The design of this merge operation additionally maintains/repairs the chain connectivity.

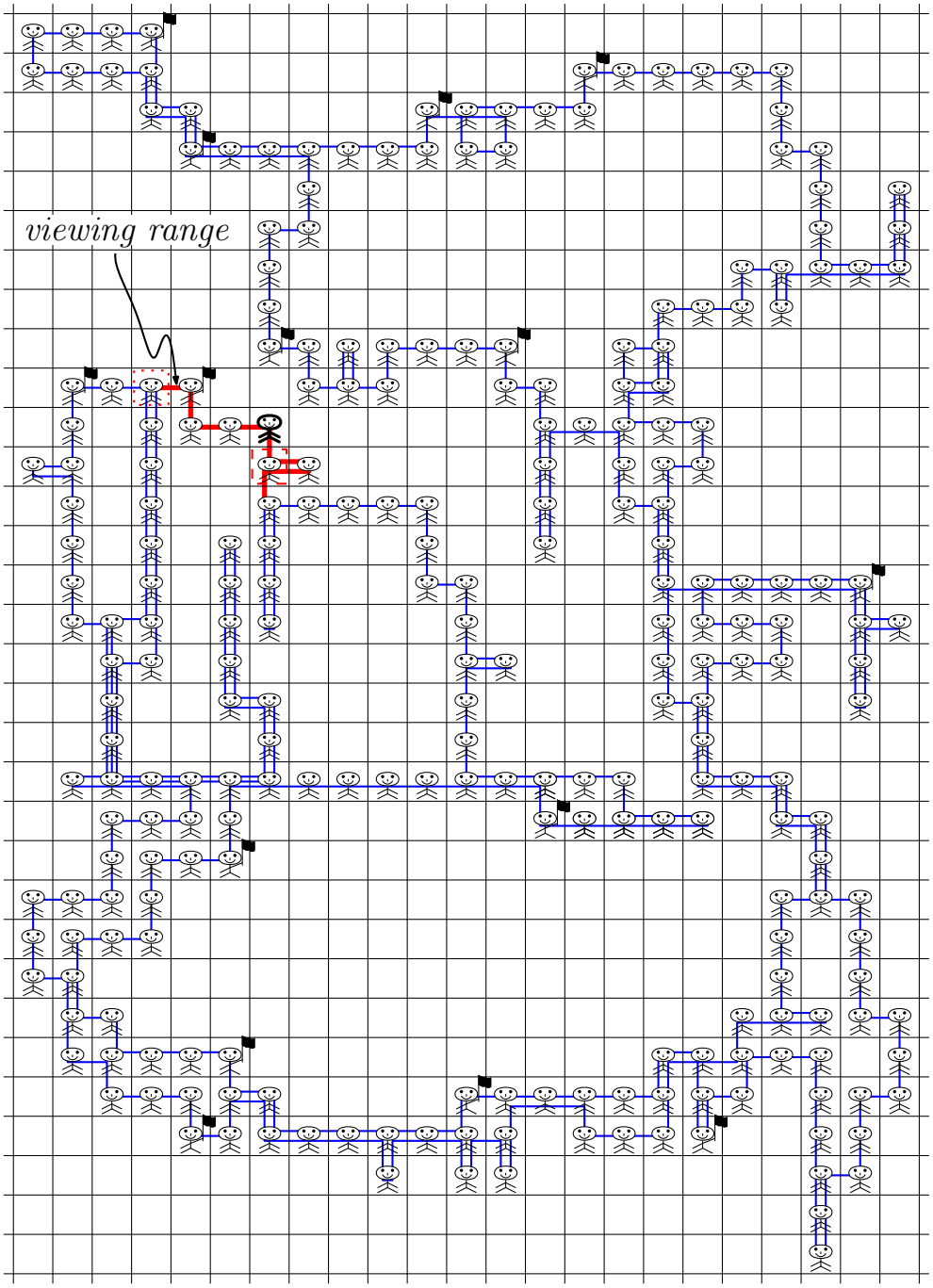


Figure 2.3: Closed-Chain-Gather: A robot swarm. The fat robot calculates its actions only on the basis of the robot positions and raised flags in its chain restricted viewing range (snapshot of the environment) that is drawn in red color. Because there is no global compass and no global chirality, the strategy must work under any rotation and reflection of the snapshot. In this example, for sake of better visualisation, we assume a viewing path length of 4.

2.2.6 Summarized robot models

Model properties	Oblivious-Gather Chapter 4	Grid-Gather Chapter 5	Closed-Chain-Gather Chapter 6
no global vision	✓	✓	✓
no sense of global coordinates	✓	✓	✓
no global compass	✓	✓	✓
no global chirality sense	✓	✓	✓
no global control (autonomous)	✓	✓	✓
indistinguishable (anonymous)	✓	✓	✓
no communication	✓	✓	✓
no memory (oblivious)	✓	–	–
no (visible) states	✓	–	–
grid neighborhood connectivity	✓	✓	–
chain neighborhood connectivity	–	–	✓
local vision (up to constant distance (*viewing range*) on the grid)	✓	✓	–
local vision (up to constant distance (*viewing path length*) along the chain)	–	–	✓
viewing range/viewing path length (L_1-distance)	7	19	11
constant-sized memory	–	✓	✓
constant number of locally visible states	–	✓	✓
hop to one of the 8 neighboring grid cells	✓	✓	✓
merge with other robots at same grid position	✓	✓	–
merge with direct chain neighbors at same grid position	–	–	✓
set/unset its own visible states	–	✓	✓
total running time	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$

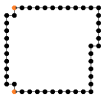
All our strategies work in the $\mathcal{FSYNC}$ time model.

CHAPTER 3

Related Work

There is vast literature on robot formation problems, researching how specific coordination problems can be solved by a swarm of robots given a certain limited set of abilities. Usually, the robots are point-shaped (hence collisions are neglected) and positioned in the Euclidean plane. They can be equipped with a memory or are *oblivious*, i.e., the robots do not remember anything from the past and perform their actions only on their current views. If robots are anonymous, they do not carry any IDs and cannot be distinguished by their neighbors. Another type of constraint is the compass model: If all robots have the same coordinate system, some tasks are easier to solve than if all robots' coordinate systems are distorted. In [Kat+07; Izu+12] a classification of these two models and also of dynamically changing compass models is considered, as well as their effects regarding the gathering problem in the Euclidean plane.

Time models. The operation of a robot is considered in the *look-compute-move* model [CP04]. We call this *LCM cycles*. Here, robot operations are split into three steps: In the *look* step, a robot takes a snapshot of the environment from its own perspective. Afterwards, in the *compute* step, it computes its action on the basis of the snapshot. Finally, in the *move* step, the calculated action is



executed.

Time can be synchronized in many different ways: In the asynchronous $\mathcal{ASYNC}$ model ([Cie+03; CP04; Flo+05; Flo+14]), a robot can decide to become inactive for a certain amount of time. Starting times of LCM cycles of active robots are also not synchronous. Furthermore, the length of the LCM cycle steps can vary individually for each robot. The length is only known to be finite and can change over time. In general, the only guarantee is that every robot becomes activated infinitely often.

A more synchronized time model is the semi-synchronous $\mathcal{SSYNC}$ model [SY99]. There, robots synchronously follow a global clock of equally sized time intervals. But robots can still be inactive. The case where all robots are always active is given in the fully synchronous $\mathcal{FSYNC}$ model [ASY95; Dyn+06; SY99; AP06]. In the synchronous time models, reasonable running time estimations are possible by counting the number of time intervals or LCM cycles [Dyn+06; Bra+13; KM09; Deg+11].

Robot shapes and movement models. Robots can have an extent or can be point-shaped. In contrast to point-shaped robots, robots with an extent in general cannot share the same position.

In the research area of so-called self-organizing particle systems, an amoeba inspired model is used, where robots have certain additional capabilities [Der+16; Der+14; Dol+13]: In the so-called *geometric amoebot model*, particles can be located on the nodes of a triangular grid, and every particle can occupy one or two nodes (if they are connected by an edge), and every node can be occupied by at most one particle. If a particle occupies only one node, it is called *contracted*, else they say it is *expanded*. It moves by alternatingly expanding and contracting. Particles on adjacent nodes are *connected*. If a particle p moves, connectivity is maintained as follows: (1) p expands to some node v that is occupied by a currently expanded particle q, then it forces ("pushes") q to contract to leave v. (2) p contracts and frees some node v', and forces ("pulls") a neighboring contracted particle q' to expand to v. The particles are anonymous, do not have a common compass, but have a common sense of chirality. They have a constant memory with a shared part that can be read and written by neighboring particles. A particle calculates its actions on the basis of the contents of its own memory and the shared memories of its local

neighbors.

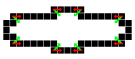
The time model is asynchronous in such a way that only one particle is active at a time [Der+16]. In the amoeba model of [Dol+13], particles are located on a hexagonal grid and have a hexagonal shape.

A different model of robots with an extent is used in [Cor+11; Sha+17], where robots are interpreted as circles.

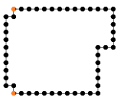
Formations of point-shaped robots. While [Der+16; Der+15] use the amoeba inspired model for shape formations, we now return to the more simple robot model, where point-shaped robots can only hop to neighboring grid points, instead of moving by contraction and expansion. Referring to such robots, the authors of [LM14] present a strategy for building almost arbitrary formations of point-shaped robots on a two-dimensional grid. Their strategy requires that every robot knows its absolute position on the grid.

In the Euclidean plane, the circle formation is a common problem [DK02; CMN04; Flo+14]. Under the restriction that robots must be very simple (no global control, no common coordinate system, no IDs, no direct communication, no memory of the past (oblivious)), Flocchini et al. prove that the uniform circle formation problem is (deterministically) solvable for any initial formation [Flo+14]. Here, uniform means that in the final formation all n robots must be located at the vertices of some regular n-gon.

One of the most natural formation problems is the gathering, where all robots have to move to the same, previously unknown point. In the related rendezvous problem only two robots have to meet or "gather" [Pel12; KKR06]. A problem, very close to gathering, is *convergence*. In contrast to gathering, where all robots have to exactly move to the same position, in convergence the robots only need to become arbitrarily close to a position.



A collection of recent algorithmic results concerning distributed solving of basic problems like gathering, convergence and pattern formation by using robots with very limited capabilities, can be found in the book of Flocchini et al. [FPS12].



Byzantine, faulty robots. The role of multiplicity detection. Concerning faulty and byzantine robots, Défago et al. [Déf+16] provide a very good comprehensive collection of feasibility analysis of the gathering task under many

relevant time models for deterministic as well as probabilistic algorithms. They analyze many variants of faulty and byzantine robots and point out the impact of the multiplicity detection robot capability. In contrast to faulty robots that just do not work, byzantine robots behave adversarially concerning the desired task. *Multiplicity detection* means that a robot can detect how many other robots are located at a position. Else, it just can detect if it is occupied or not [DP12; Izu+09]. Also *weak* and *strong* multiplicity detection can be distinguished [Izu+09]. Here, weak means that a robot can only distinguish if there is no, one or more than one robot located at a position (but not the actual number).

Communication Chains

One basic robot formation problem is the shortening and maintenance of a communication chain between two fixed endpoints. The robots then only have a local vision, no compass and a static connectivity, defined by neighborhoods along the chain. For this problem several results have been published in the $\mathcal{FSYNC}$ time model. The first shown runtime bound was $\mathcal{O}(n^2 \log(n))$ [Dyn+06]. Later, this has been improved [KM09]: In the Euclidean plane, the *Hopper* strategy delivers a $\sqrt{2}$-approximation of the shortest communication chain in time $\mathcal{O}(n)$. Restricted to a grid, the *Manhattan Hopper* strategy delivers an optimal solution in time $\mathcal{O}(n)$.

Gathering

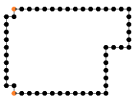
A very basic natural problem is the already mentioned gathering, where all robots have to gather at a single, previously undefined point. There, the swarm usually consists of point-shaped, oblivious, and anonymous robots. The problem is widely studied in the Euclidean plane. Having point-shaped robots, collisions are understood as merges/fusions of robots and interpreted as gathering progress [DKM10]. In [Cie+03] the first gathering algorithm for the $\mathcal{ASYNC}$ time model with multiplicity detection (i.e., a robot can detect if other robots are also located at its own position) and global views is provided. Gathering in the local setting was researched in [ASY95]. Degener et al. [Deg+11] have shown a $\mathcal{O}(n^2)$ running time bound for a local gathering strategy Go-To-The-Center in $\mathcal{FSYNC}$ that works in the Euclidean plane. Furthermore, they

showed that for their strategy this bound is tight. In [Pre07] situations when no gathering is possible are studied. The question of gathering on graphs instead of gathering in the plane was considered in [Mar09; Des+06; KMP08]. In [SN13] the authors assume global vision, the $\mathcal{ASYNC}$ time model and furthermore allow unbounded (finite) movements. They show optimal bounds concerning the number of robot movements for special graph topologies like trees and rings.

Gathering on a grid

Concerning the gathering on grids, in [DAn+12] it is shown that multiplicity detection is not needed and the authors further provide a characterization of solvable gathering configurations on finite grids. In [SN14], these results are extended to infinite grids, assuming global vision. The authors characterize *gatherable* grid configurations concerning exact gathering in a single point. Under their robot model and the $\mathcal{ASYNC}$ time model, the authors present an algorithm which gathers *gatherable* configurations optimally concerning the total number of movements.

Local gathering. Assuming only local capabilities of the robots, especially only local vision and no compass, makes gathering challenging. For example, given a global vision, the robots could compute the center of the globally smallest enclosing square or circle and just move to this point. Concerning gathering with presence of a global compass, the authors in [SMM16] provide a simple gathering algorithm for robots on a grid that needs only time $\mathcal{O}(n)$: In their strategy, the robots from the left and right swarm boundaries keep moving towards the swarm's inside. In some kinds of degenerated cases, robots from the top and bottom boundaries do this instead. But without the presence of a global compass, certain arbitrarily scalable swarm formations cannot be gathered using their strategy: e.g., swarms in vertically aligned zigzag shapes do not work then.



Local gathering in $\mathcal{FSYNC}$. In the $\mathcal{FSYNC}$ time model, the total running time is a quality measurement of an algorithm. In this time model and additionally under the restriction that the robots do not have a compass and have

only local vision, i.e., they can only see other robots up to a constant distance instead of a global vision of the whole scenery, there exist several results that prove runtime bounds.

One of these are the above introduced strategies for shortening communication chains: The strategy of Dynia et al. [Dyn+06] needs time $\mathcal{O}(n^2 \log(n))$. The one of Kutylowski et al. [KM09] needs time $\mathcal{O}(n)$ in the Euclidean plane as well as on a grid. The Closed-Chain-Gather strategy in this thesis can be seen as a transition from these open communication chain shortening strategies to the gathering problem, because it performs the gathering by shortening the closed chain that connects all robots [Abs+16] (Chapter 6 of this thesis). For this strategy, we have proven an asymptotically optimal running time bound of $\mathcal{O}(n)$.

For the more general gathering, i.e., using connectivity dynamically given by local vision in the Euclidean plane, an $\mathcal{O}(n^2)$ runtime bound has been shown by Degener et al. [Deg+11] for their Go-To-The-Center strategy, as already mentioned above. There, every robot synchronously computes the smallest enclosing circle only of the robots within its restricted viewing range and then moves towards its center. Repeatedly executing this synchronous behavior finally solves the gathering. The authors also prove that for their algorithm the $\mathcal{O}(n^2)$ bound is tight. For the problem itself, under this local model, a tight bound for the running time is still unknown. In [Deg+11], Degener et al. use a very minimalistic robot model: Besides the strictly local capabilities, their robots do not have a memory (oblivious) and do not have visible states, lights or flags. In our paper [FJM17a] (Chapter 4 of this thesis), we use the same minimalistic robot model, but on the grid. We prove a $\mathcal{O}(|\text{outer boundary}|^2) \subseteq \mathcal{O}(n^2)$ upper running time bound for this Oblivious-Gather strategy. This matches the above one of Degener et al. [Deg+11]. Our simulations (cf. Section 4.7) let us assume that at least for our strategy this upper bound is also tight. In our paper [Cor+16] (Chapter 5 of this thesis), we tightened this bound to the asymptotically optimal value of $\mathcal{O}(n)$. For this, we extended the robot model by a constant-sized memory for remembering a constant number of previous actions and a constant number of states that are locally visible to other robots.

CHAPTER 4

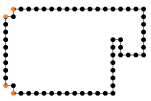
Gathering Anonymous, Oblivious Robots on a Grid

Oblivious-Gather Strategy

Basic&Plain Robot Model

We consider the results from Degener et al. [Deg+11]. The authors showed for a very minimalistic local robot model that gathering in the Euclidean plane can be solved in time $\mathcal{O}(n^2)$. Their robot model matches our *Basic&Plain* robot model. The authors' gathering strategy Go-To-The-Center works in the fully synchronous $\mathcal{FSYNC}$ time model.

In the current chapter, we contribute the (to the best of our knowledge) first gathering algorithm on the grid that works under the time model and the same minimalistic robot model as this Euclidean plane strategy. We prove its correctness and an $\mathcal{O}(n^2)$ running time bound in the fully synchronous $\mathcal{FSYNC}$ time model. This time bound matches the time bound of the best known algorithm for the Euclidean plane mentioned above. We say gathering is done if all robots are located within a 2×2 square, because in $\mathcal{FSYNC}$ such configurations cannot be solved.



4.1 Introduction

A basic desired global behavior of a swarm is the gathering process: All robots have to gather at one (not predefined) place. Local algorithms for this problem are defined and analyzed for a variety of models [Deg+11; DKM10; FPS12; Cie+03; Cor+16; Abs+16].

A common local model for extremely simple robots is the following: There is no global coordinate system. The robots have no common compass and no global sense of chirality, only have a constant viewing radius, are autonomous and indistinguishable, can move at most a constant distance in each step, cannot communicate, are fully oblivious and do not have flags or lights to show a state to others. In this very restricted robot model, a robot's decision about its next action can only be based on the current relative positions of the otherwise indistinguishable robots in its constant-sized viewing range, and independent of past decisions or information (oblivious). This is the model that we have called *Basic&Plain* in the introduction of this thesis.

The only gathering algorithm (Go-To-The-Center) under this robot model, with known runtime bounds, needs $\mathcal{O}(n^2)$ rounds and works in the Euclidean plane [Deg+11]. The underlying time model for the algorithm is the fully synchronous $\mathcal{FSYNC}$ model.

In the discretization of the Euclidean plane, the two-dimensional grid, no running time bounds for gathering are known under the same time and robot model. The concept of the Euclidean Go-To-The-Center algorithm from [Deg+11] furthermore cannot be transferred to the grid: In Figure 4.1, the fat robot computes its next action of the Go-To-The-Center strategy. It can see only the robots within the fat circle (viewing range). Then, it computes the minimum enclosing circle of itself and these robots. Afterwards, it moves towards this circle's center point. This strategy does not work on the grid, because firstly it must be possible to move to the exact position of the minimum enclosing circle center point and secondly a robot must be able to move arbitrary short distances. Both clearly is impossible on the grid. Instead, completely different approaches are needed.

In our new Oblivious-Gather strategy, a robot compares the relative positions of the robots in its viewing range to several patterns. On the basis of the matching pattern, it decides to hop onto a certain neighboring grid cell or stay

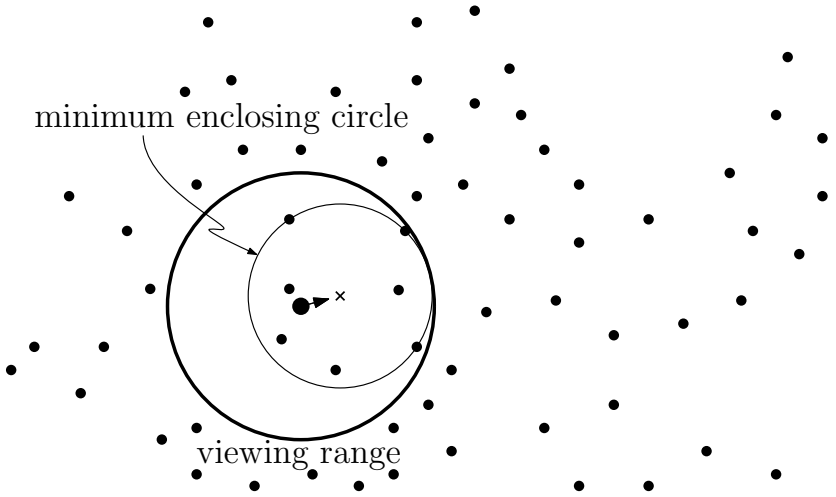


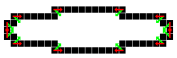
Figure 4.1: Euclidean *Go-To-The-Center* strategy: Every robot calculates its action on the basis of the smallest enclosing circle that includes itself and all robots in its viewing range.

at its current position. For an example, see Figure 2.1 on page 13.

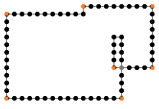
Using this procedure in our model makes ensuring a reasonable behavior hard for the following reasons:

1. Under the lack of knowledge of any global positions, a global compass and a global chirality, the locally compared patterns must induce a reasonable behavior under every rotation and mirroring.

2. On the grid, a constant viewing radius also means that a robot can see only a fixed constant number of cells and robots, independent of the global swarm size and progress of the gathering.



3. The absence of memory (oblivious) implies that if a robot action leads to gathering progress, then this progress must be achieved immediately after the action is executed, because in the next LCM cycle the robots do not remember their previous action.



The strategy itself is relatively easy to explain, but formally proving the correctness and a total running time bound turned out to be difficult. We need to combine three different kinds of progress measures: Roughly speaking, we measure the length of the swarm's outer boundary, count the number of

convex vertices on the outer boundary and measure the area that is covered by the swarm.

To the best of our knowledge, we present the first local strategy under the restricted *Basic&Plain* model, i.e., without memory (oblivious) and without visible states, on the grid and prove a total running time of $\mathcal{O}(n^2)$ rounds which complies with the best known running time for the Euclidean strategies in this model [Deg+11]. More precisely, the running time of our strategy depends quadratically on the outer boundary length of the swarm. The outer boundary is the seamless sequence of neighboring robots that encloses all the others robots inside.

We conjecture that $\Omega(n^2)$ is a lower bound for the number of rounds needed for our algorithm and, more generally, even for any algorithm within our restricted model. At least for our algorithm, we conjecture that a worst case instance is a configuration with robots on the boundary of an axis-parallel square. Experiments support this conjecture (cf. Section 4.7).

4.2 Our Local Model

Our mobile robots need very few and simple capabilities: A robot moves on a two-dimensional grid and can change its position to one of its eight horizontal, vertical or diagonal neighboring grid cells. It can see other robots only within a constant *viewing radius* of 7 (measured in L_1-distance). We call the snapshot of visible robot positions the *viewing range*. Within this viewing range, a robot can see only the relative positions of the viewable robots. The robots have no compass, no global control, and no IDs. They cannot communicate, do not have any states and are oblivious.

Our algorithm uses the fully synchronous time model $\mathcal{FSYNC}$, in which all robots are always active and do everything synchronously.

The robots simultaneously execute their operations in the common *look-compute-move model* [CP04], which divides one operation into three steps. Every round contains only one cycle of these steps: In the *look* step, a robot gets a snapshot of the current scenario from its own perspective, restricted to its constant-sized viewing range. During the *compute* step, the robot computes its action, and eventually performs it in the *move* step. We say it executes Look-Compute-Move or LCM cycles.

In the current Oblivious-Gather strategy, robot operations are very simple. More precisely, they are finished after a single LCM cycle. We call this LCM cycle a *round* that has a constant length. In order to estimate the running time, we simply have to count these rounds.

If a robot has moved to an occupied grid cell, the robots from then on behave like one robot. We say they *merge* and remove one of them.

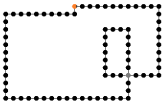
We say gathering is done if all robots are located within a 2×2 square, because such configurations cannot be solved in our time model.

The swarm must be connected. In our model, two robots are connected if they are located in horizontal or vertical neighboring grid cells. The operations of our algorithm do not destroy this connectivity.

4.3 The Algorithm

A robot decides to hop on one of its 8 neighboring grid cells only dependent on the current robot positions within its viewing range. We distinguish diagonal (Subsection 4.3.1) and horizontal/vertical (Subsection 4.3.2) hops. The hops are intended to achieve the gathering progress by modifying the swarm's *outer boundary*. Figure 4.6.(i) defines the swarm's boundaries: Black and hatched robots are *boundary* robots. The boundary on which the black robots are located borders the swarm and is called the swarm's *outer boundary*. In the figure, all other robots are colored grey. White cells are empty.

4.3.1 Diagonal hops



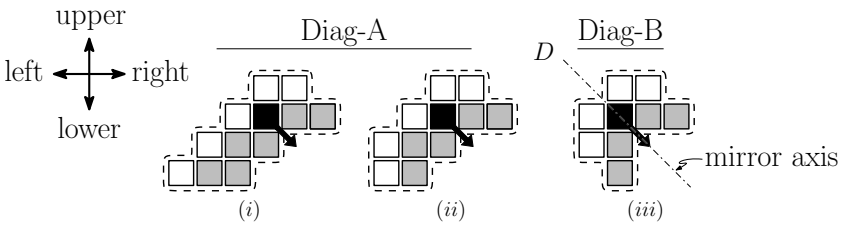


Figure 4.2: **Hop patterns:** One of the Diag-A or Diag-B Hop patterns must match the relative robot positions within the black robot's viewing range. This is the hop criterion, necessary for allowing the black robot to perform the depicted diagonal hop. In this chapter, the notation Diag-{A, B} means "Diag-A or Diag-B".

If a robot r (marked black in Figures 4.2 and 4.3) checks whether it can execute a diagonal hop, it compares the patterns in Figure 4.2 and 4.3 to the robot positions in its viewing range: Robot r checks if one of the Diag-{A, B} Hop patterns matches the current scenario from its own perspective. Patterns that are created by an arbitrary horizontal and vertical mirroring and an arbitrary 90 degree rotation of the three patterns in Figure 4.2 are also valid and have to be checked.

Depending on the matching Hop pattern, the robot does the following:

1. If a Diag-A pattern matches, then robot r checks if, using the same rotation and mirroring, any of the Inhibit patterns (cf. Figure 4.3) match the *upper-right area* of its viewing range. If at least one Inhibit pattern matches, then the Diag-A hop of robot r is not executed. Otherwise, if none of the Inhibit patterns match, the robot r hops according to the matching Diag-A pattern.

2. If the Diag-B pattern matches, then the robot r checks if, using the same rotation and mirroring, any of the Inhibit patterns match the *upper-right area* or the *lower-left area* of its viewing range. However, in case of the *lower-left area*, the Inhibit pattern has to be mirrored at the diagonal mirroring axis D shown in Figure 4.2.(iii). If for both areas matching Inhibit patterns have been found, then the Diag-B hop of robot r is not executed. Otherwise the robot r hops according to the matching Diag-B pattern.

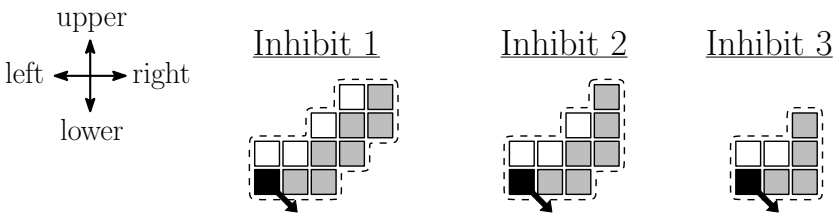


Figure 4.3: **Inhibit patterns:** Patterns that, in case they match, inhibit the black robot's hop.

4.3.2 Horizontal and vertical hops (HV hops)

Robots can also hop in vertical or horizontal direction (HV hop). We allow these hops for length 1 and 2 (cf. Figure 4.4). For length 2, horizontal or vertical hops, respectively, are a joint operation of two neighbouring robots. If for a robot a horizontal and a vertical HV hop apply at the same time (see $b^\star$ of Fig. 4.4), then it instead performs a diagonal hop as shown in the figure. After

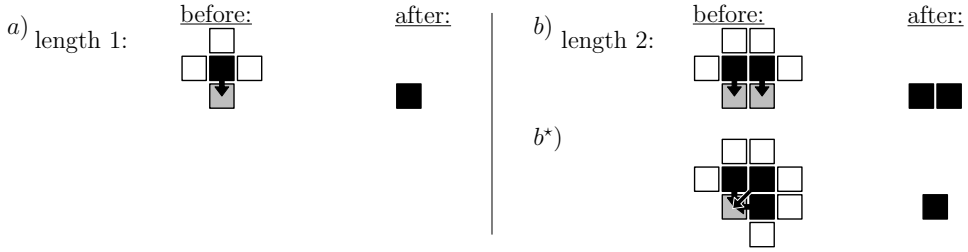


Figure 4.4: The black robots simultaneously hop downwards. Afterwards, robots that are located at the same position *merge*.

a HV hop, every target cell contains at least two robots. We let these robots *merge*, i.e., we remove all but one of the robots from the according cell.

Diag-{A, B} and HV hops are processed simultaneously by the robots. In summary, all robots synchronously execute the algorithm, shown in Figure 4.5.

Every robot r in every round does the following:	
Look:	r takes a snapshot of the relative robot positions within its viewing range.
Compute:	r checks which of the HV hop, Diag-{A, B} hop and Inhibit patterns match the snapshot. On the basis of the matching patterns, r decides if it either does not hop or on which neighboring grid cell it will hop.
Move:	If r has calculated a hop, it is now executed. Else, r does not move.

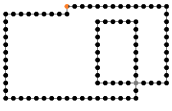


Figure 4.5: The algorithm.

4.4 Measuring the Gathering Progress

The gathering progress measures that we will use for the analysis of our strategy are heavily dependent on the length and shape of the swarm's outer boundary. In order to analyze these measures, we need to define the terms boundary, outer boundary, length, as well as convex and concave vertices.

Swarm's boundary. The swarm's boundary is the set of all robots that have at least one empty adjacent cell in a horizontal, vertical, or diagonal direction. Figure 4.6.(i) shows an example: Black and hatched robots are boundary robots. The empty cells contain no robot and are colored in white. When speaking about a *subboundary*, we mean a connected sequence of robots of some boundary.

Swarm's outer boundary. The swarm's outer boundary is the boundary that borders on the outside of the swarm. In Figure 4.6.(i) the black robots belong to the outer boundary. All other robots are not part of the boundaries, i.e., they have an adjacent robot in all directions (horizontal, vertical, and diagonal). In Figure 4.6.(i) they are colored grey.

Outer boundary's length. We measure the outer boundary's length as follows: We start at a cell of the outer boundary and perform a complete walk along this boundary, while we define the *length* as the total number of steps that we performed during this walk. This means that if the swarm is hourglass- or cross-shaped, for example, some robots are counted multiple (up to four) times (cf. Figure 4.6.(iii, iv)). Furthermore, robots on which a turn by $\pm 180°$ is performed during the walk are counted twice (cf. Figure 4.6.(v)). We denote this length by |outer boundary|.

Convex and concave vertices. On the boundary we further distinguish *convex* and *concave vertices*. Depending on the relative positions of its neighbor robots, a robot is called a convex/concave vertex: In Figure 4.6.(ii) fat curves mark *convex vertices* of the swarm's outer boundary, while the thin curves mark the *concave vertices* of the swarm's outer boundary.

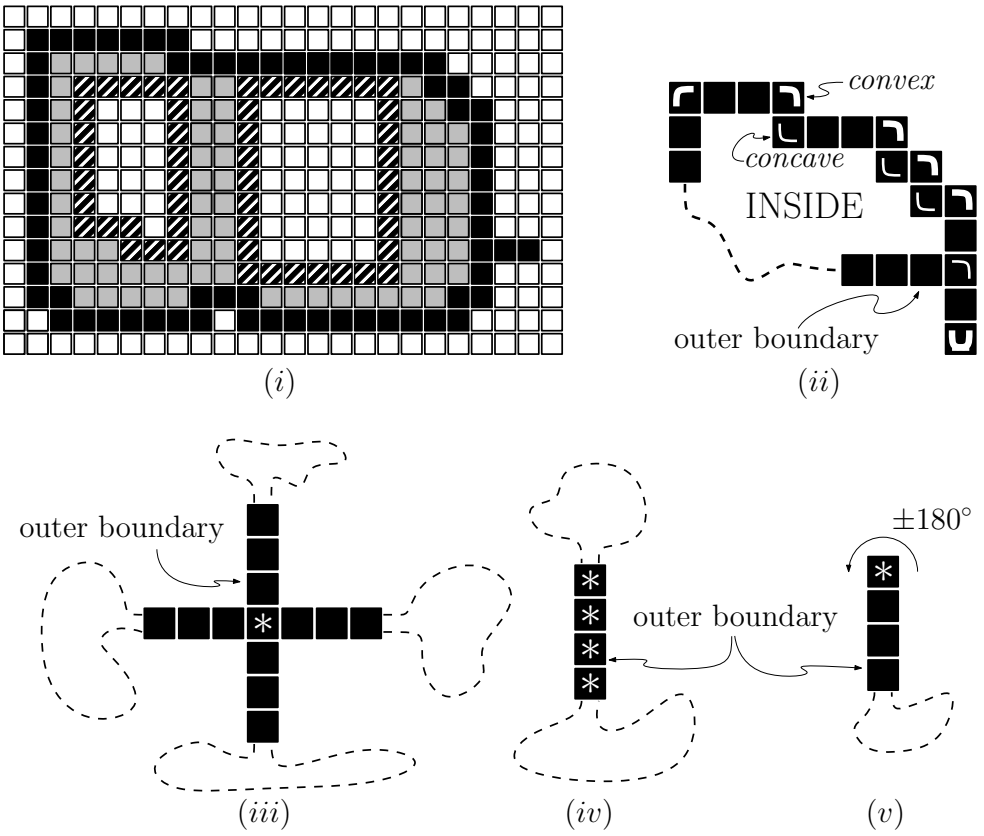


Figure 4.6: (i): Definition of the (*outer*) *boundary*. Outer boundary: black robots; (ii): Definition of *convex* and *concave vertex*. Convex vertices: fat curves; (iii): Cross shape. The "$*$" marked robot is counted four times; (iv): Hourglass shape: The "$*$" marked robots are counted twice; (v): $\pm 180°$ rotation. The "$*$" marked robot is counted twice.

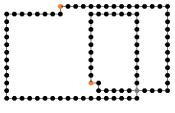
Outline of the running time proof — How we achieve gathering progress.

We distinguish three kinds of progress measures that help us to prove the quadratic running time.

***Boundary*:** Length of the swarm's outer boundary.

***Convex*:** Difference between the number of convex vertices on the swarm's outer boundary and its maximum value.

***Area*:** Included area.



We have designed the hops in such a way that the length of the outer boundary (*Boundary*) never increases. But it can remain unchanged over several rounds.

Then, instead, we measure the progress by *Convex*. As we draw robots as squares, the total number of convex vertices on the swarms' outer boundary is naturally upper bounded by its maximum value $4 \cdot |\text{outer boundary}|$. *Convex* is the difference between this maximum value and the actual number of convex vertices on the outer boundary. We will show that also *Convex* never increases.

In rounds in which both *Boundary* and *Convex* do not achieve progress, we instead measure the gathering progress by *Area*. We measure *Area* as the number of robots on the swarm's outer boundary plus the number of inside cells (occupied as well as empty ones). In contrast to the other progress measures, *Area* does not decrease monotonically in general, but we show that it decreases monotonically in rounds without *Boundary* and *Convex* progress. We upper bound the size by that the *Area* can instead be increased during other rounds and show that this makes the total running time worse by at most a constant factor.

All three measures depend only on the length of the swarm's outer boundary. While *Boundary* and *Convex* are linear, *Area* is quadratic.

This then leads us to a total running time $\mathcal{O}(|\text{outer boundary}|^2)$.

Section 4.8 shows an example that visualizes which of the three measures have progress during the stepwise gathering of an empty square.

4.5 Correctness & Running Time

In this section, we formally prove the correctness of the progress measures and finally the total running time (Theorem 4.4).

4.5.1 Progress measure *Boundary*

Lemma 4.1. *During the whole gathering, Boundary is monotonically decreasing.*

Proof. As the definition of HV hops requires that the robots hop onto occupied cells, such hops naturally cannot increase the number of robots on the outer boundary. So we consider Diag-$\{A, B\}$ hops in which robots hop towards the swarm's outside. In order to increase the number of robots on the outer boundary, the target cell of such hops must be empty. But then, the hopped robot has also been part of the outer boundary before the hop, so that the boundary length did not increase. □

4.5.2 Impact of Inhibit patterns: *Collisions*

For the analysis of the progress measures *Convex* and *Area*, we need a deeper insight why certain robot hops are inhibited by Inhibit patterns (cf. Figure 4.3). When proving that *Convex* progress is monotonically decreasing (Lemma 4.2), we analyze the change of the total number of convex vertices that is induced by the robot hops. This is possible only if certain simultaneously hopping robots are not too close together. Inhibit patterns ensure this minimum distance.

Cf. Figure 4.2 and 4.3. From a more global point of view, the Inhibit patterns ensure that the black robot only performs its Diag-A or Diag-B hop, respectively, if the next robot(s) at distance 2 along the boundary does (do) not perform a hop in the opposite direction. If the hop of the black robot is blocked by Inhibit patterns, we will say the robots *collide*. This will be used in the proofs of our progress measures. Figure 4.7 shows significant collision examples: (i): For both, r and r' Diag-A matches. Without inhibition patterns, r would hop to the lower right, while r' would hop in the opposite direction to the upper left. But the Inhibit 1 pattern inhibits the hop of r: r collides with r'. Analogously also the hop of r' can be inhibited. (ii): If Diag-B matches for r, then the hop is inhibited if concerning both r' and r'' an inhibition pattern matches. In this example, for r' a Diag-A pattern and for r'' a Diag-B pattern could else enable hops in the opposite direction than the hop of r, but the matching Inhibit 1 and 2 patterns inhibit the hop of r: r collides with r', r''. A more detailed analysis of collisions is provided in the proofs for Lemma 4.3.

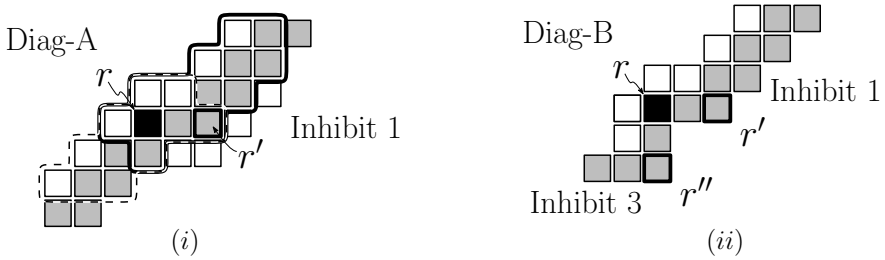


Figure 4.7: *Collisions*. (i): For r, Diag-A and Inhibit 1 matches. r does not hop. (ii): For r, Diag-B and Inhibit 1 and 3 match. r does not hop.

4.5.3 Progress measure *Convex*

Lemma 4.2. *During the whole gathering, Convex is monotonically decreasing.*

Proof. If we say "convex/concave vertices", we consider only the outer boundary. First, we analyze the HV hops. Here, a HV hop can reduce the number of convex vertices by at most 2. At the same time, the outer boundary becomes shorter by at least 2. Then, *Convex* either remains unchanged or decreases.

Concerning the Diag-{A, B} hops, we look at Figure 4.8: The figure shows

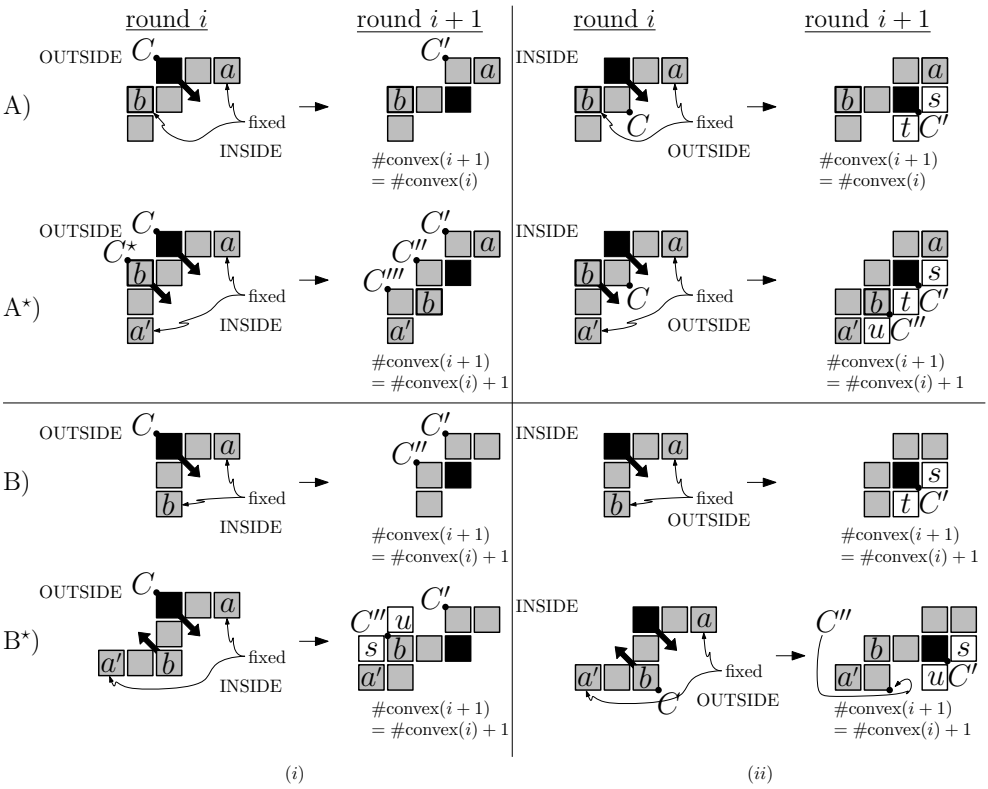


Figure 4.8: Local effect of all kinds of hops on the number of convex vertices on the outer boundary. C, C', C'' denote the counted convex vertices. #convex(i) denotes the number of convex vertices in round i.

how the diagonal hops can (locally) change the number of convex vertices on the outer boundary. (In the figure, (ii) shows the hops from (i), but for switched INSIDE and OUTSIDE.) In all cases, the Inhibit patterns ensure that the robots a, a' do not move. In the figure, we distinguish Diag-A and Diag-B hops, where A, A* refer to Diag-A and B, B* to Diag-B. We distinguish the case that robot b does not hop (A, B) and the other case, that it performs a hop (A*, B*). The result of the case distinction is that in column (i) the number

of convex vertices never decreases. In column (ii), this is also the case if the white marked cells s, t, u are empty. If instead not all of s, t, u are empty, the number of convex vertices might also become smaller. But even in this case, *Convex* progress still does not increase: We now show that then also |outer boundary| becomes smaller as well as the maximum value $4 \cdot$ |outer boundary| for the total number of convex vertices, so that by definition *Convex* progress is not increased, i.e., it still behaves monotonically. Figure 4.9 shows the relevant

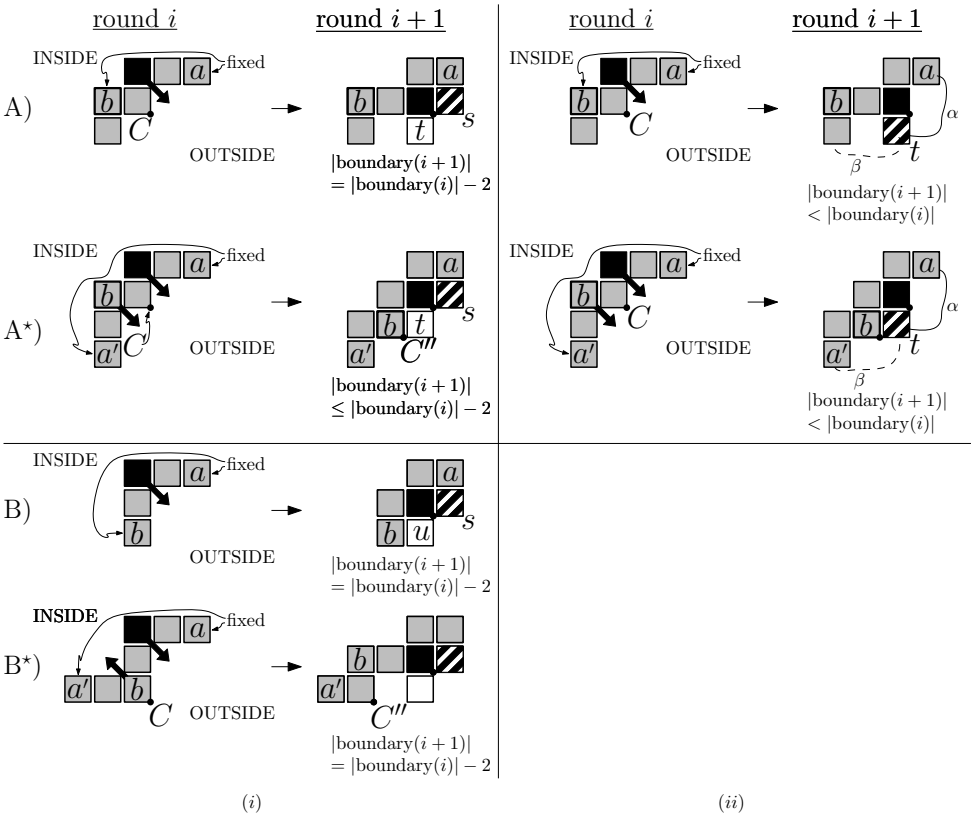


Figure 4.9: Diagonal hops can change the outer boundary's length. C, C', C'' denote relevant convex vertices. #boundary(i) denotes the outer boundary length in round i.

cases. With reference to Figure 4.8.(ii), all hops are performed towards the swarm's outside. In column (i), only the cell s contains a robot. There we see that in all cases the outer boundary becomes shorter by at least 2. In case of A*, this can be even more than 2 for the case that after the hop the cell below b (in

Figure 4.8.(ii), this was the cell u) also contains a robot. Column (ii) shows the cases where cell t is (also) occupied. Because the swarm is always connected, the hatched robot must be connected to the rest of the swarm. This can be either via the subboundary α or β. The hop shortens the outer boundary by forming inner bubbles (Figure 4.10). □

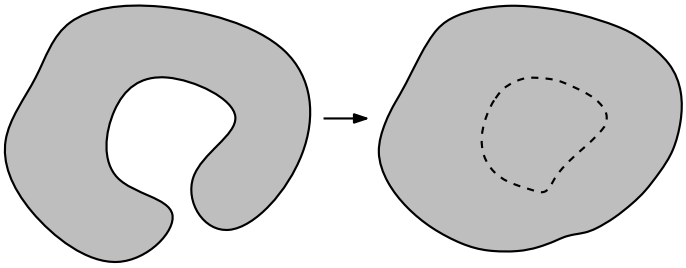


Figure 4.10: During the gathering, inner bubbles can be developed.

4.5.4 Progress measure *Area*

The third progress measure *Area* does not behave monotonically. It can be increased during rounds where we get *Boundary* or *Convex* progress. But we use it for estimating the number of the remaining rounds (Lemma 4.3). And in the proof of Theorem 4.4 we show that the increased amount of the *Area* progress measure does not increase the asymptotic running time.

Lemma 4.3. *If in a step of the gathering process neither Boundary nor Convex has progress, then instead Area has progress by at least* -8.

For the proof of Lemma 4.3, see Section 4.6.

4.5.5 Total running time

Now we can combine all three progress measures *Boundary*, *Convex* and *Area* for the running time proof (Theorem 4.4).

Theorem 4.4. *A connected swarm of n robots on a grid can be gathered in* $\mathcal{O}(|\text{outer boundary}|^2) \subseteq \mathcal{O}(n^2)$ *many rounds.*

Proof. Let B be the initial length of the swarm's outer boundary. We know from Lemma 4.1 that *Boundary* decreases monotonically. Then, progress in *Boundary*

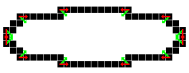
happens at most B times. By Lemma 4.2, *Convex* also decreases monotonically. As every robot on the swarm's outer boundary can provide at most 4 convex vertices, *Convex* progress happens at most $4B$ times.

We estimate the rounds without *Boundary* and *Convex* progress via the size of the included area, i.e., the *Area* progress. By Lemma 4.3, we know that in every round without *Boundary* and *Convex* progress, the area becomes smaller by at least -8. However, *Area* is not a monotone progress measure, in rounds with *Boundary* or *Convex* progress the included area can increase: While HV hops cannot increase the included area, Diag-{A, B} hops can. First, we assume that the according Diag-{A, B} hops do not change the outer boundary length. Then, every time *Convex* has progress, the area can become larger by at most B, because every robot hop on the outer boundary can increase the area by at most 1. As *Convex* happens at most $4B$ times, this in total is upper bounded by $4B^2$.

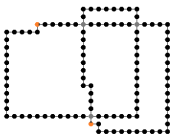
If the outer boundary length changes, i.e., becomes shorter, then the included area can increase (cf. proof of Lemma 4.2 and Figure 4.10). Then, a *Boundary* progress by ℓ can also increase the included area by $\Delta A \leq \ell^2 \leq \ell B$. But as *Boundary* progress is monotonically decreasing, the sum of all these ΔA is upper bounded by B^2.

Summing it up, during the whole process of the gathering, the area can be increased by at most $(4+1)B^2 = 5B^2$. Together with the initial area of at most B^2, *Area* progress happens at most $6B^2$. Then, the gathering is done after at most $B + 4B + 6B^2$ rounds. □

4.6 Proof of Lemma 4.3



In this section, we provide the proof of Lemma 4.3. In the following, we will prove Lemma 4.7 and Lemma 4.8 whose combination immediately deliver the proof of Lemma 4.3. Lemma 4.7 and 4.8 are given later, when we have prepared for their proofs.



4.6.1 Outline of the proof

In Lemma 4.7, we assume that the swarm does not contain hourglass-shaped parts, while in Lemma 4.8 the result is generalized to swarms that do contain

hourglass-shaped parts. In Figure 4.17, hourglass-shaped parts are marked by black robots. We call them *bridges*. A more detailed definition is given before Lemma 4.7.

For the proofs, we will first construct a more abstract model of the swarm's outer boundary: During a walk along the swarm's outer boundary one performs rotations by $0°$ or $\pm 90°$. We subdivide the outer boundary into parts (so-called *supercorners*) such that at both of their endpoints a Diag-$\{A, B\}$ hop can be performed and that during a walk along such a supercorner, a total rotation by $0°$ or $\pm 90°$ is performed. We will argue that because a total rotation by $+360°$ is always performed during a complete counter-clockwise walk along the swarm's outer boundary, there must be 4 more $+90°$ than $-90°$ supercorners. In our construction, robot hops of $+90°$ supercorners are always performed towards the swarm's inside. This then proves the statement of Lemma 4.7 that *Area* decreases by at least -8.

When generalizing this to Lemma 4.8, we first notice that bridges can hinder hops towards the swarm's inside. But we will argue that a single bridge cannot hinder the total -8 *Area* decrease that is proven by Lemma 4.7, and that at least two bridges are needed for this. We subdivide the swarm into bridges and subswarms that are connected by bridges, as shown in Figure 4.17 on page 49. Then, we construct a tree such that every node represents a bridgeless subswarm and every edge represents a bridge. The leaves of this tree represent subswarms that only contain one bridge. The proof of Lemma 4.8 then mainly follows by the argument that every tree contains enough leaves for compensating the non-decreasing *Area* progress of the subswarms that are represented by its inner nodes.

4.6.2 Preparing for the proofs

Our arguments mainly consider only robots on the swarm's outer boundary. However, in general, a swarm also contains inside robots. The proofs of Lemma 4.7 and 4.8 require that inside robots neither inhibit outer boundary robots from hopping towards the swarm's inside (*Area* decrease) nor enable additional robot hops towards the swarm's outside (*Area* increase). We formally prove this in Lemma 4.5.

Lemma 4.5. *Let ∂S be the outer boundary of some swarm S and S° be its interior robots. We consider Diag-{A, B} hops:*

1. *If $r \in \partial S$ hops towards the swarm's inside, it also does this on S.*
2. *If $r \in \partial S$ does not hop towards the swarm's outside, it also does not hop on S.*

Roughly speaking, in the proof we analyze how Diag-{A, B} and Inhibit patterns can be generated or destroyed by adding robots and prove that in situations in which this would lead to a contradiction of the lemma, this cannot be done by adding robots only to the swarm's interior. For the formal proof, see Subsection 4.6.6.

For the *Area* progress proofs, we need a more formal description of the swarm's outer boundary.

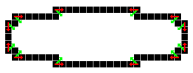
Quasi lines.

Definition 4.6 (quasi line)**.** We define a subboundary, called a horizontal quasi line, as follows (cf. Figure 4.11):

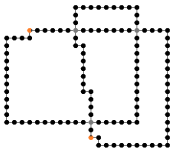
1. It consists only of horizontal subboundaries of length ≥ 3 that are connected by *stairways* of height 1 or 2. (Definition of stairways, see below.)
2. It begins and ends with three horizontally aligned robots.

In Figure 4.11, *stairways* are marked by black robots. They are alternating left and right turns. In the figure, their endpoints are marked by bicolored robots.

Stairways that do not match the quasi line Definition 4.6, i.e., are of height > 2, connect two neighboring quasi lines that can be either both horizontal, both vertical, or one horizontal and the other one vertical. Our algorithm performs Diag-{A, B} hops at these connection points. In Figure 4.11, dashed lines border the according patterns for *quasi line* 1.



In our proofs, we represent a horizontal quasi line by a horizontal line segment and a stairway by a diagonal one. For the example in Figure 4.11, the drawn polygon shows this construction for the given example swarm.



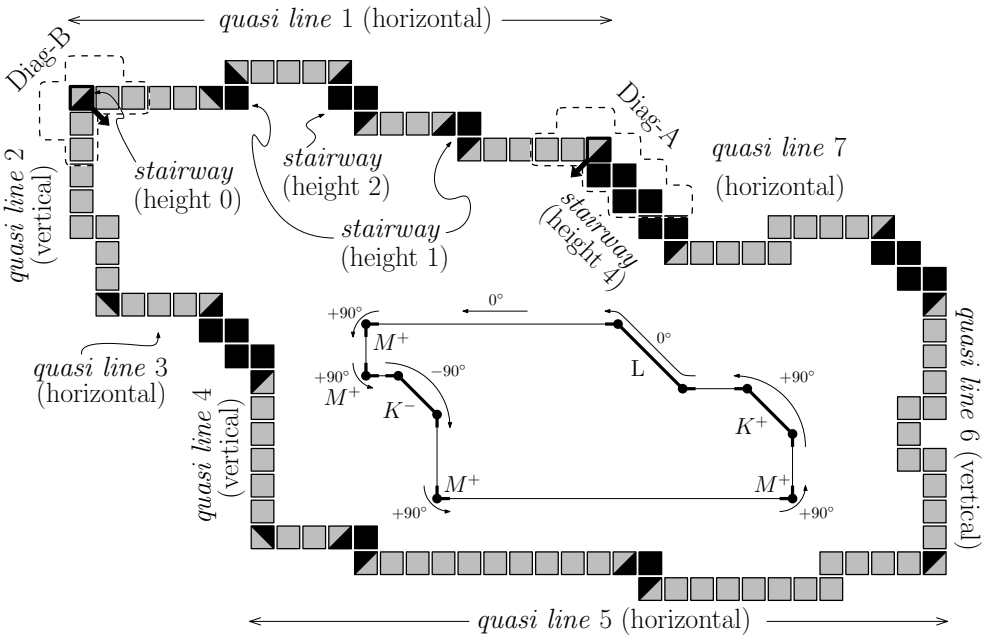


Figure 4.11: Example: Quasi lines and stairways and an abstract representation of the shown boundary.

Corners. We call the connections between different quasi lines *corners*. These are the fat drawn parts of the polygon. At their endpoints, the Diag-{A, B} Hop patterns match. The formal definition of *corners* is the following (cf. Figure 4.12).

K^+ Corner includes a diagonal line segment and is convex (i.e., induces a +90° rotation).

K^- Corner includes a diagonal line segment and is concave (i.e., induces a −90° rotation).

L Corner includes a diagonal line segment and one end is convex and the other one concave (i.e., induces 0 rotation).

M^+ Corner does not include a diagonal line segment and is convex (i.e., induces a +90° rotation).

M^- Corner does not include a diagonal line segment and is concave (i.e., induces a −90° rotation).

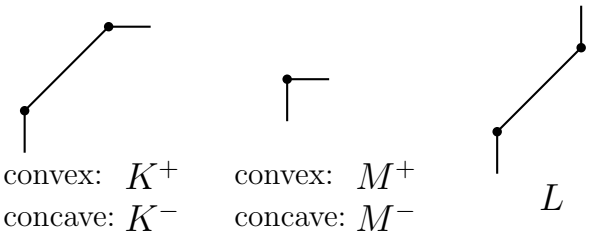


Figure 4.12: Definition of *corners*.

Supercorners. In our strategy, though the visible pattern of some robot matches one of Diag-{A, B}, it does not execute its hop if suitable inhibition patterns match. Remember the difference that if for a robot a Diag-A pattern matches, only one matching inhibition pattern inhibits its hop, but for Diag-B two (one in each direction) inhibition patterns are required. Figure 4.13 shows

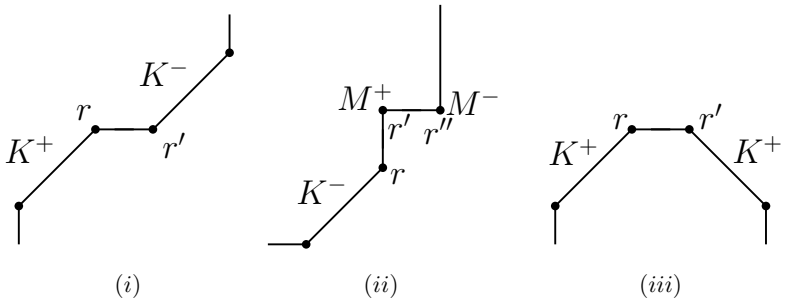
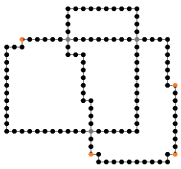


Figure 4.13: Collisions in corner representation. (Significant examples)

significant examples for this: (i): r, r' do not hop. (ii): r, r' do not hop, but the robot r'' does, because a colliding robot exists only in one direction on the boundary (Diag-B pattern). (iii): $r,' r$ execute their hops, because they both hop downwards (no collision).

Now we construct *supercorners* $\mathcal{X}, \mathcal{Y}, \mathcal{Z}$ from consecutive K^+, K^-, L, M^+, M^- corners in such a way that the supercorners have the property that only the two (respectively the one for supercorners that only consist of a single $M^{\pm}$ corner) endpoint robots actually perform their hops and the hops of all other included K^+, K^-, L, M^+, M^- endpoint robots are inhibited by collisions with neighbors. Because the hops must be performed in opposite directions in order to collide, the consecutive corners K, M along $\mathcal{X}, \mathcal{Y}, \mathcal{Z}$ must be alternately convex and



concave. This means that we can define the supercorners analogously to corners (cf. Figure 4.14). Here, n_{K^+}, n_{M^+} denotes the number of K^+, respectively M^+ corners of the according supercorner and n_{K^-}, n_{M^-} denotes the same but for K^- and M^-, respectively.

$\mathcal{X}^+$: $n_{K^+} + n_{M^+} = n_{K^-} + n_{M^-} + 1$ (total $+90°$ rotation)
$\mathcal{X}^-$: $n_{K^+} + n_{M^+} = n_{K^-} + n_{M^-} - 1$ (total $-90°$ rotation)
$\mathcal{Y}$: $n_{K^+} + n_{M^+} = n_{K^-} + n_{M^-}$ (no total rotation)
$\mathcal{Z}^+$: $n_{M^+} = 1; n_{K^+}, n_{K^-}, n_{M^-} = 0$ ($+90°$ rotation)
$\mathcal{Z}^-$: $n_{M^-} = 1; n_{K^+}, n_{K^-}, n_{M^+} = 0$ ($-90°$ rotation)

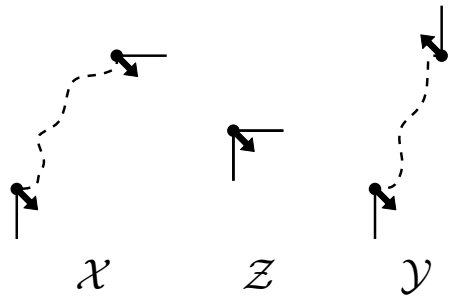


Figure 4.14: Definition of *supercorners* $\mathcal{X}, \mathcal{Y}, \mathcal{Z}$.

4.6.3 *Bridges*

In Lemma 4.7 and 4.8, we distinguish two kinds of swarms: the ones without (Lemma 4.7) and the ones with *bridges* (Lemma 4.8). For the definition of *bridges*, we start with a swarm (cf. Figure 4.15.(i)) which has the property that it contains exactly two endpoints (hatched) from which the whole swarm can be completely removed just by continuously executing HV hops. Figure 4.15.(ii): We call such a subswarm a *bridge* if it connects two other subswarms (grey) at its endpoints.

4.6.4 *Area* progress for swarms without bridges (Lemma 4.7).

Lemma 4.7. *If a swarm S does not contain any bridges and neither Boundary nor Convex has progress, then instead Area has progress.*

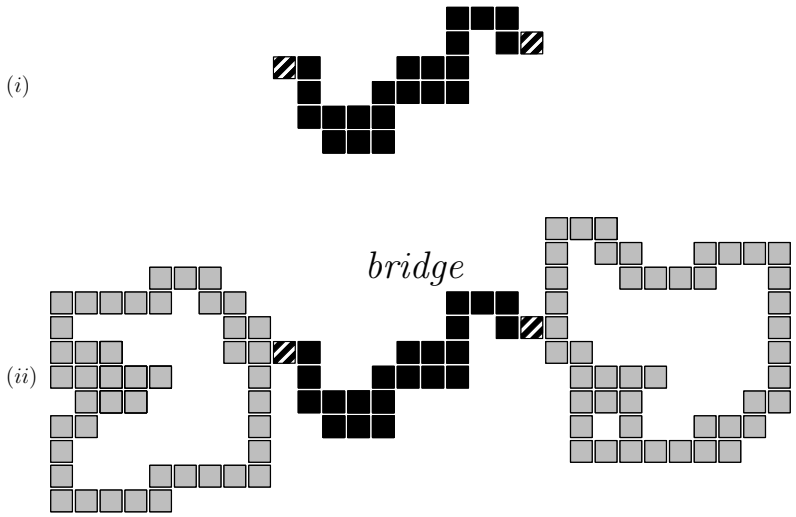
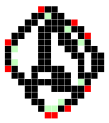


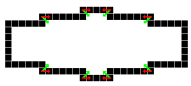
Figure 4.15: (i): The black subswarm can be completely removed by executing a sequence of HV hops from its two endpoints (hatched). (ii): The *bridge* connects two subswarms.

Proof. During a walk along the outer boundary of the swarm, we perform a total rotation of 360°. Using the above construction, it follows:



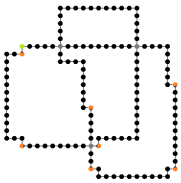
$$N_{K^+} + N_{M^+} = N_{K^-} + N_{M^-} + 4,$$

where $N_{K^+}, N_{M^+}, N_{K^-}, N_{M^-}$ are the total numbers of K^+, M^+, K^- and M^-, respectively. Because of the difference +4, there must exist neighboring corners without collisions. Then, by construction, the above equation also holds for supercorners:



$$N_{\mathcal{X}^+} + N_{\mathcal{Z}^+} = N_{\mathcal{X}^-} + N_{\mathcal{Z}^-} + 4.$$

By the prerequisites of the lemma, there is no *Convex* progress. So, $N_{\mathcal{Z}^+} \stackrel{!}{=} 0$. Note that $N_{\mathcal{Z}^-}$ can be bigger than zero if the according hop is hindered by inside robots. We get:



$$N_{\mathcal{X}^+} = N_{\mathcal{X}^-} + N_{\mathcal{Z}^-} + 4.$$

By construction, every $\mathcal{X}^+$ executes two hops towards the swarm's inside, while $\mathcal{X}^-$ does the same towards the outside. Then, the included area behaves

as follows:

$$\Delta Area = -2(N_{\mathcal{X}^+} - N_{\mathcal{X}^-}) = -2(N_{\mathcal{Z}^-} + 4) \leq -8.$$

This finishes the proof. □

4.6.5 *Area* progress for swarms with bridges (Lemma 4.8).

Now we generalize Lemma 4.7 to a swarm that includes *bridges*.

Lemma 4.8. *If a swarm $\mathcal{S}$ contains $k > 0$ bridges and neither Boundary nor Convex has progress, then instead Area has progress.*

Proof. We classify subswarms by the number of bridges by which they are connected to the rest of the swarm.

#bridges $= 1$ (leaves). We first look at a subswarm S' which is connected by only one bridge B to the remaining part of $\mathcal{S}$. Later, we will call a subswarm with this property a *leaf*. For our analysis, we separate S' from $\mathcal{S}$ by splitting B into two parts (e.g., this could be done by removing ≤ 2 robots). Afterwards, we remove the part that is then connected to S' by executing a sufficient number of HV hops. The remaining subswarm $\tilde{S}'$ from S' then only allows Diag-{A, B} hops. Now we start with the same construction as in the proof of Lemma 4.7, but this time applied only to the subswarm $\tilde{S}'$. Accordingly, we end up in the equation:

$$N^{\tilde{S}'}_{\mathcal{X}^+} + N^{\tilde{S}'}_{\mathcal{Z}^+} = N^{\tilde{S}'}_{\mathcal{X}^-} + N^{\tilde{S}'}_{\mathcal{Z}^-} + 4.$$

In contrast to the proof of Lemma 4.7, here $N^{\tilde{S}'}_{\mathcal{Z}^+}$ can be bigger than 0, in case B hinders the hops of all existing $\mathcal{Z}^+$ supercorners.

Now we estimate the worst case for the number of diagonal hops towards the inside of $\tilde{S}'$ that could have been hindered by the bridge B. There are two cases how B can hinder a diagonal hop:

1. It occupies white marked cells in the Diag-{A, B} patterns.
2. It produces a collision with a robot that wants to perform a Diag-{A, B} hop.

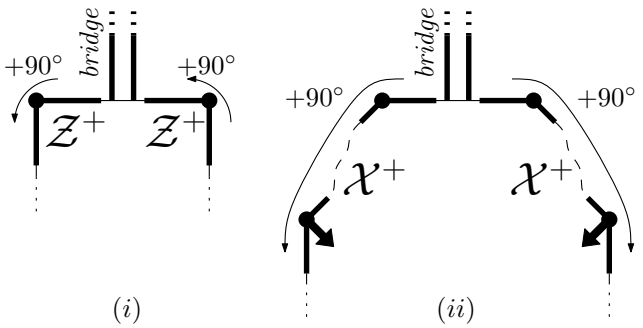


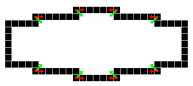
Figure 4.16: A bridge hinders at most two hops towards the swarm's inside.

As the width of B by definition is ≤ 2, it can hinder at most two hops towards the inside of $\tilde{S}'$. In Figure 4.16, we see that the $\mathcal{Z}^+$ supercorners ((i)) provide the worst case, because here we can hinder *all* hops of two convex +90° supercorners. If, e.g., hops of $\mathcal{X}^+$ supercorners are hindered instead ((ii)), then afterwards still two of their robots hop towards the swarm's inside.

For the worst case, we get the equation

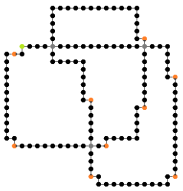
$$\begin{aligned} N^{\tilde{S}'}_{\mathcal{X}^+} + N^{\tilde{S}'}_{\mathcal{Z}^+} &= N^{\tilde{S}'}_{\mathcal{X}^-} + N^{\tilde{S}'}_{\mathcal{Z}^-} + 4 \\ N^{\tilde{S}'}_{\mathcal{X}^+} + 2 &= N^{\tilde{S}'}_{\mathcal{X}^-} + N^{\tilde{S}'}_{\mathcal{Z}^-} + 4 \\ N^{\tilde{S}'}_{\mathcal{X}^+} &= N^{\tilde{S}'}_{\mathcal{X}^-} + N^{\tilde{S}'}_{\mathcal{Z}^-} + 2. \end{aligned}$$

By construction, every $\mathcal{X}^+$ executes two hops towards the swarm's inside, while $\mathcal{X}^-$ does the same towards the outside. As we, because of the prerequisites of the lemma, do not have *Convex* progress, all $\mathcal{Z}^-$ hops are hindered by inside robots (i.e., $N^{\tilde{S}'}_{\mathcal{Z}^-}$ might be > 0.). Then the included area behaves as:



$$\Delta Area^{S'} \leq -2(N^{\tilde{S}'}_{\mathcal{X}^+} - N^{\tilde{S}'}_{\mathcal{X}^-}) = -2(N^{\tilde{S}'}_{\mathcal{Z}^-} + 2) \leq -4.$$

I.e., in $\tilde{S}'$ respectively S', the included area becomes smaller by at least 4 instead 8 for swarms without bridges.



#bridges ≥ 1. We can generalize this worst case construction to a subswarm $S^\star$ that is connected by $k \geq 1$ bridges to the remaining swarm. For this, analogously to the one-bridge-case, we first build $\tilde{S}^\star$ from $S^\star$ by removing the

bridges.

There, every bridge can still hinder at most 2 hops towards the subswarm's inside and, as before, the worst case means that these are $\mathcal{Z}^+$ corners. We get $N^{\tilde{S}^\star}_{\mathcal{Z}^+} = 2k$. So, in total

$$\begin{aligned} N^{\tilde{S}^\star}_{\mathcal{X}^+} + 2k &= N^{\tilde{S}^\star}_{\mathcal{X}^-} + N^{\tilde{S}^\star}_{\mathcal{Z}^-} + 4 \\ N^{\tilde{S}^\star}_{\mathcal{X}^+} &= N^{\tilde{S}^\star}_{\mathcal{X}^-} + N^{\tilde{S}^\star}_{\mathcal{Z}^-} + 4 - 2k \end{aligned}$$

and

$$\begin{aligned} \Delta Area^{S^\star} &\leq -2(N^{\tilde{S}^\star}_{\mathcal{X}^+} - N^{\tilde{S}^\star}_{\mathcal{X}^-}) \\ &= -2(N^{\tilde{S}^\star}_{\mathcal{Z}^-} + 4 - 2k) \\ &\overset{(*)}{\leq} -8 + 4k, \end{aligned}$$

where for $N^{\tilde{S}^\star}_{\mathcal{Z}^-} = 0$ $(*)$ becomes a strict equality (worst case).

This means that in the worst case, for 2 bridges the area does not change and for > 2 bridges it even increases. We will compensate this by showing that there are still enough subswarms that are only connected by a single bridge (leaves).

For showing this, we first transform the swarm to a tree.

Tree. Cf. Figure 4.17. We divide the swarm into bridges and subswarms S_i (in the figure, the subswarms S_i are named $A, B, \ldots, G$.). In the graph representation, every S_i becomes a node while every bridge is interpreted as an edge. The resulting graph is a tree, because in our construction an existing cycle would represent an "area", i.e., one of the subswarms S_i, and so would contradict our construction in which every S_i is a node.

Summarized *Area* progress. We estimate the total amount of area progress, by inductively constructing this tree: We start with the root node v_0, which has degree k_0. Then also k_0 leaves are connected to this node. We know that for this v_0 in the worst case *Area* behaves like $\Delta Area^{v_0} = -8 + 4k_0$, which can be ≥ 0. But adding the area change of the leaves, we get $\Delta Area \leq (-8+4k_0) - 4k_0 = -8$.

Now we inductively add the other inner nodes to the tree. Then one can show that for every tree the following holds: Every inner node v of degree

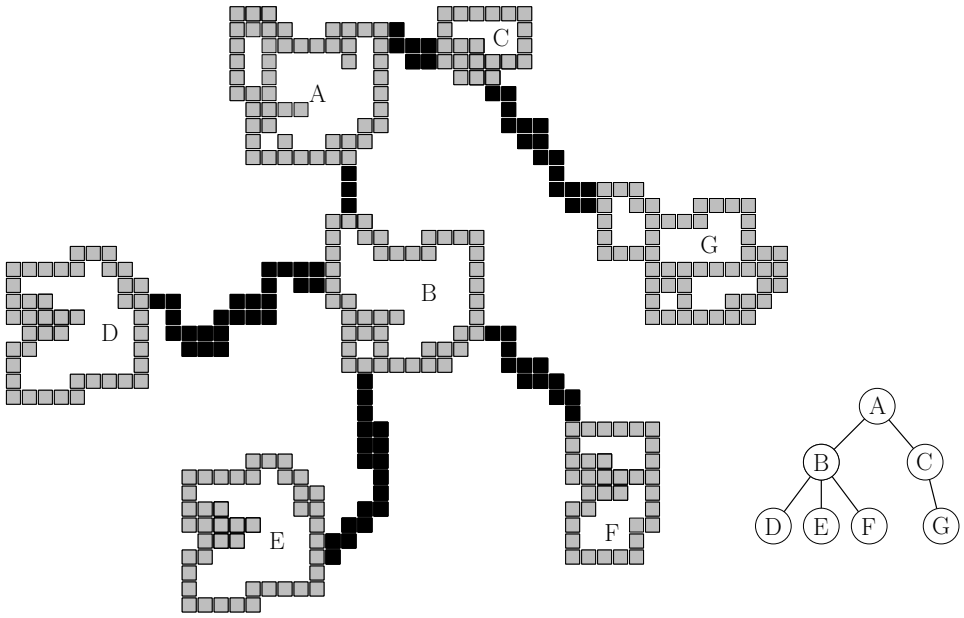


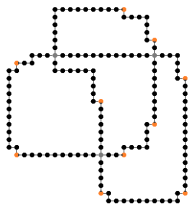
Figure 4.17: Transformation of a swarm with bridges to a tree.

k serves $k-2$ additional leaves. If we associate these leaves to v, we get $\Delta Area^v \leq (-8+4k)-4(k-2)=0$. So v does not have any bad impact on the change of total included area: $\Delta Area = \Delta Area^{v_0} + \Delta Area^v \leq -8+0=-8$. This finishes the proof. □

Lemma 4.7 and 4.8 immediately lead to Lemma 4.3.

4.6.6 Proof of Lemma 4.5

Proof. In this proof, if talking about collisions that inhibit hops, we often analyze the case when, in the viewing range of a robot, a Diag-A pattern changes to Diag-B and vice versa. This is relevant, because by definition a Diag-B hop is inhibited only if two Inhibit patterns match at the same time (one in each direction), while for Diag-A hops only one is needed. Now, we start with the proofs:



1.) The hop of r can be hindered by inside robots by two reasons. a) The Hop pattern of r changes: This is impossible, as for this outside robots must be added.

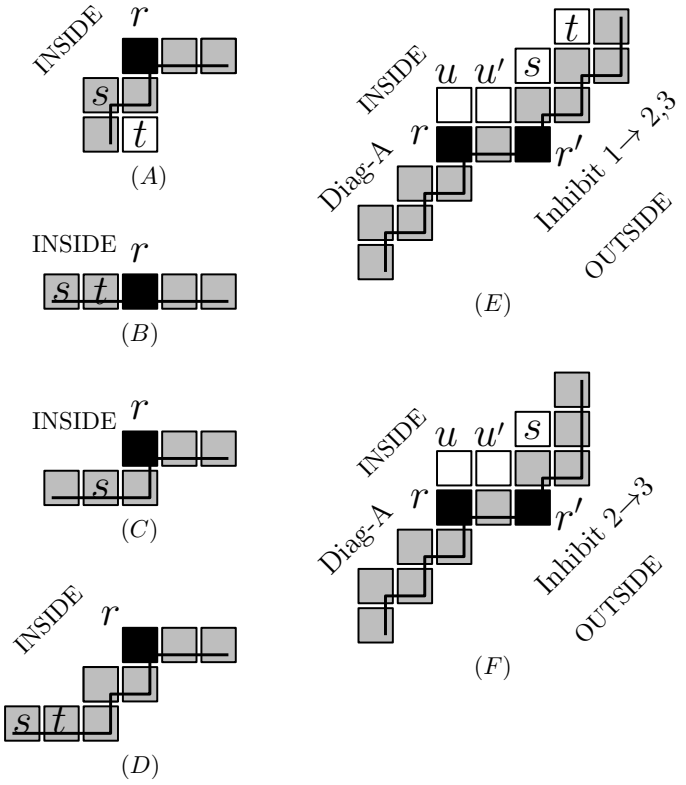


Figure 4.18: Illustrations for the proof of Lemma 4.5.

b) Collisions: (*i*): *r* changes from Diag-B to Diag-A: This is not possible, because of *a*).

(*ii*): Some robot *r*′ becomes part of an Inhibit pattern that makes him collide with *r*: This is not possible, because if *r* checks for collisions, inhibition patterns do not include inside robots.

2.) Cf. Figure 4.18. The bold polygonal line marks the outer boundary. The hop of a robot *r* that shall hop towards the swarm's outside can be enabled by inside robots by two reasons. *a*) The robot positions in the viewing range of *r* change: (*i*): *r* changes from Diag-A to Diag-B. Figure 4.18.(*A*): This is not possible, because for this, *s* must be removed and *t* occupied. Both is not possible.

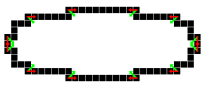
(*ii*): *r* was located on a *quasi line* and changes to Diag-{A, B}. *α*): Figure 4.18.(*B*): *r* was located on a group of horizontally aligned robots. This

would require to remove at least the robot t. β): Figure 4.18.(C): r was located on a stairway of height 1. This would require to remove at least the robot s. γ): Figure 4.18.(D): r was located at an endpoint of a stairway of height 2. This would require to remove one of the robots s, t. δ): r could also not be located inside a stairway, because then the white marked cells in the Diag-{A, B} patterns are not both free and also cannot be emptied.

b) Collisions: (i): r was Diag-A and becomes Diag-B. This is not possible, because of a).(i). (ii): Previously, r collided with some robot r' and after adding inside robots, they do not collide anymore. Figure 4.18.(E): We assume that for r a Diag-A pattern matches (The proof for Diag-B is the same.). Initially, the Inhibit 1 pattern inhibits the hop. The cells u, u' must stay empty because else the Hop pattern for r would be destroyed. But we can change the occupancy of the cells s, t: α): If we add a robot to t, while s stays empty, then the Inhibit 1 pattern becomes an Inhibit 2 pattern and the collision is still present. β): If we add a robot to s, then, independent of t, the Inhibit 3 pattern matches and the collision is still present. γ): If initially an Inhibit 2 pattern induces the collision ((F)), then adding a robot to s makes the Inhibit 2 pattern become an Inhibit 3 pattern. This covers all variants, because Inhibit 3 patterns cannot be transformed. □

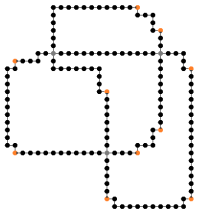
4.7 Simulation Results

We provided swarms that are shaped as empty (unfilled) squares of different sizes as an input of our Oblivious-Gather strategy. (In case of the empty square, the |outer boundary| also equals the number of robots that build the corresponding swarm.)



We count the number of rounds needed for reaching two different events:

1. Gathering is done.
2. The smallest enclosing rectangle of the swarm becomes smaller the first time.



In the plot of Figure 4.19 we have lower bounded both measures by functions that are quadratic in |outer boundary| in order to support that the simulative results of both curves are in $\Omega(|\text{outer boundary}|^2)$.

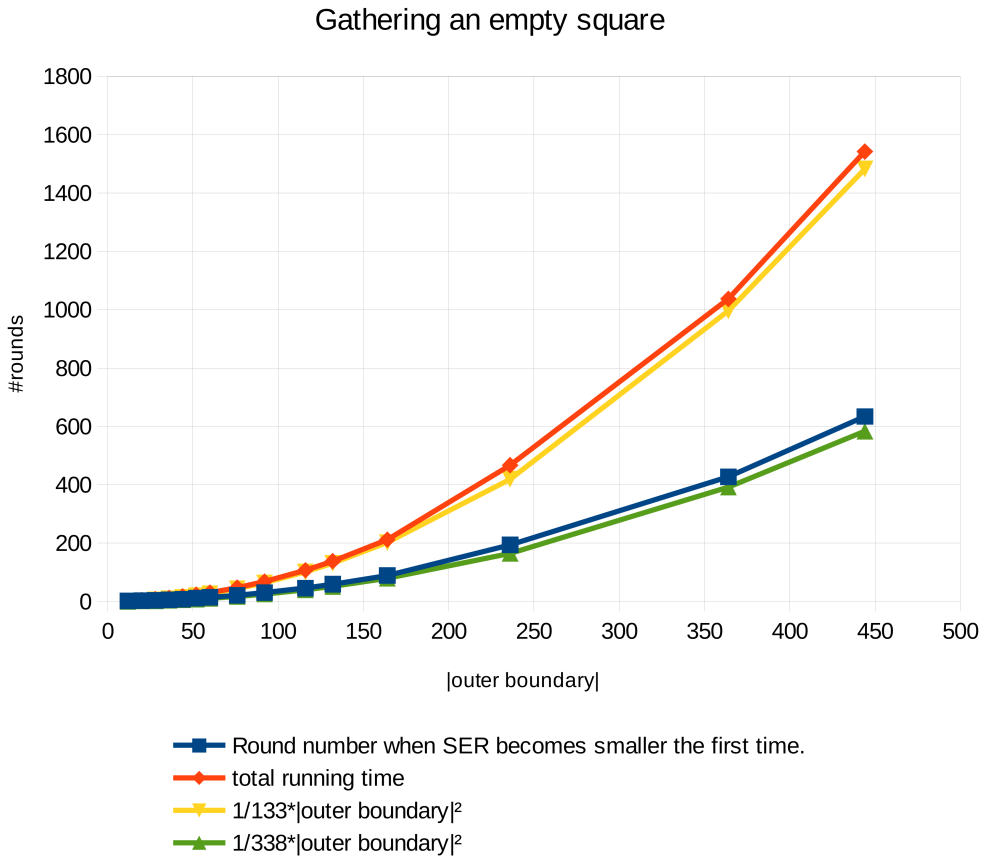


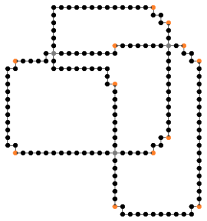
Figure 4.19: Simulation of the strategy, applied to an unfilled square, consisting of n robots.

While the plot of 1. already shows that the total running time seems to be in $\Omega(|\text{outer boundary}|^2)$, the plot of 2. is more noteworthy: The event when the smallest enclosing rectangle of the swarm becomes smaller for the first time, also marks the first occurrence of the linear *Boundary* progress. Firstly, this already seems to take quadratic time $\Omega(|\text{outer boundary}|^2)$. Secondly, reaching this event already requires about half of the total time needed for the gathering. On the swarm shapes that occur afterwards, the strategy works much faster. That is why the empty square seems to be a worst-case input for our strategy.

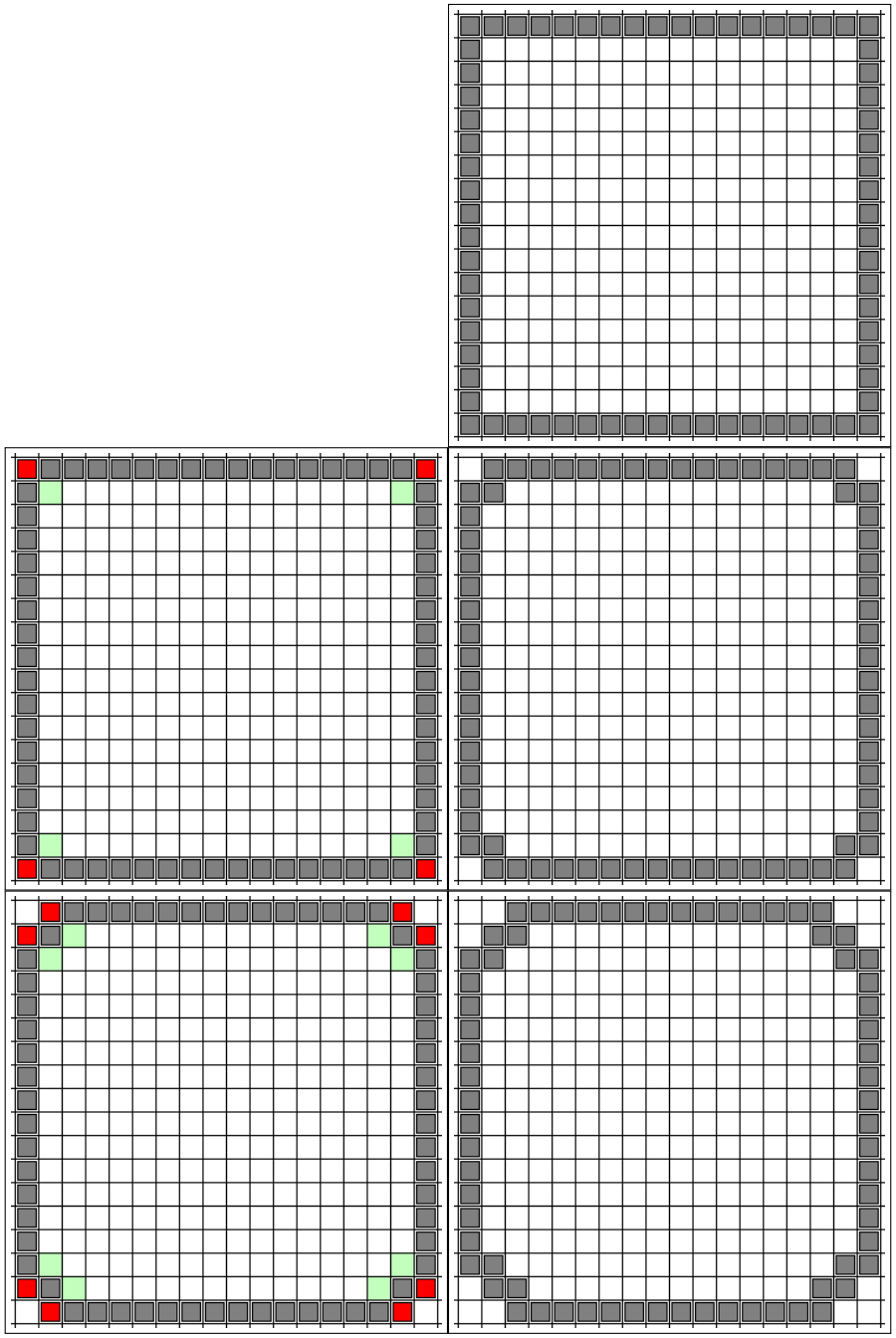
If we look at the example in Section 4.8, where an empty square consisting of 68 robots is gathered, this effect is also visualized: In round 17 the first boundary progress occurs and the smallest enclosing rectangle becomes smaller the first time. If we continue the gathering in this example, then only 20 rounds

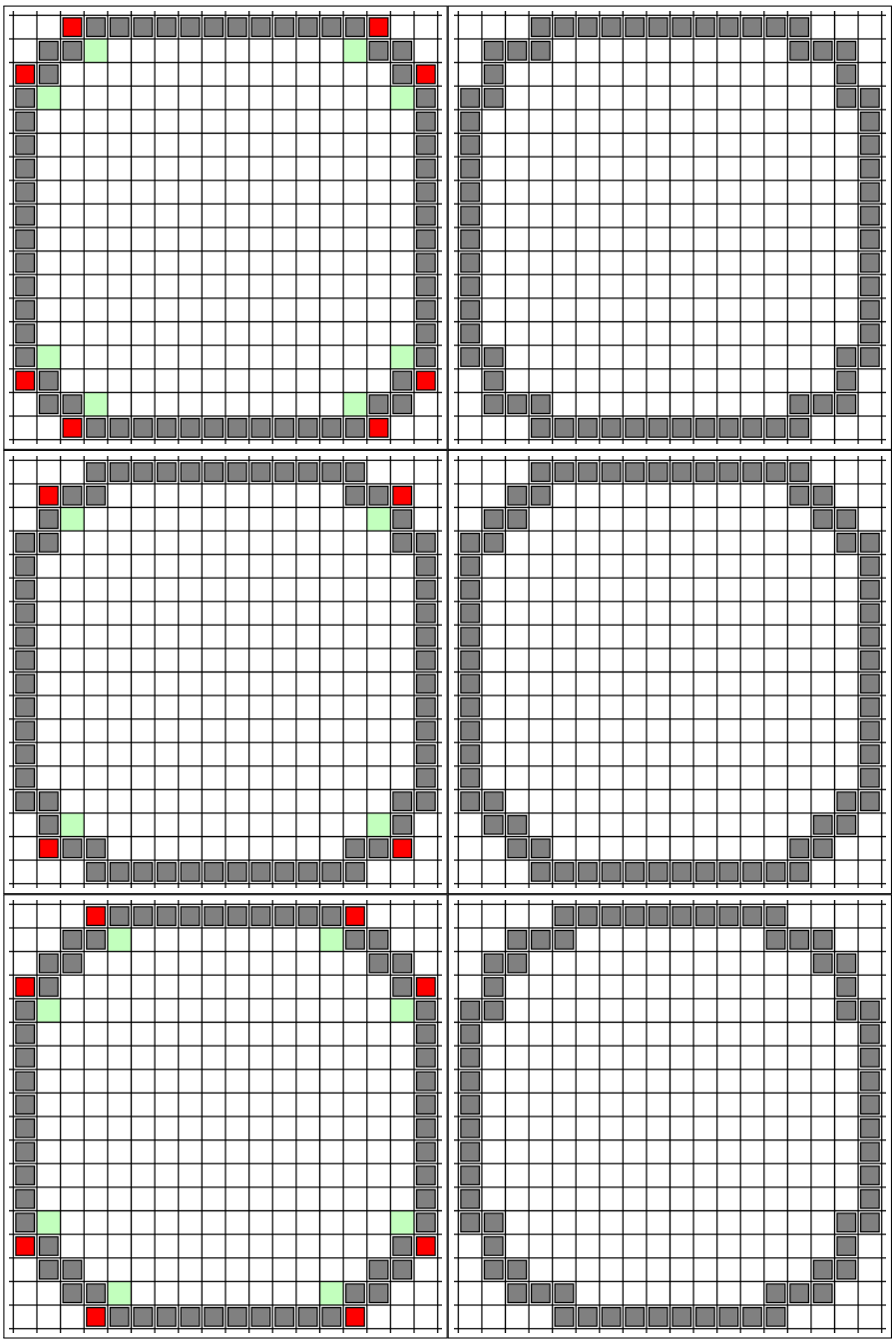
later, the whole gathering is done.

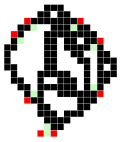
For a video animation of the gathering of a big square see also Section 1.2.2 on page 1.2.2.

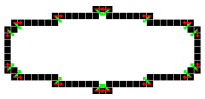


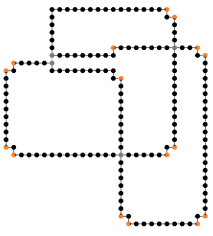
4.8 Simulation Results – Progress Analysis

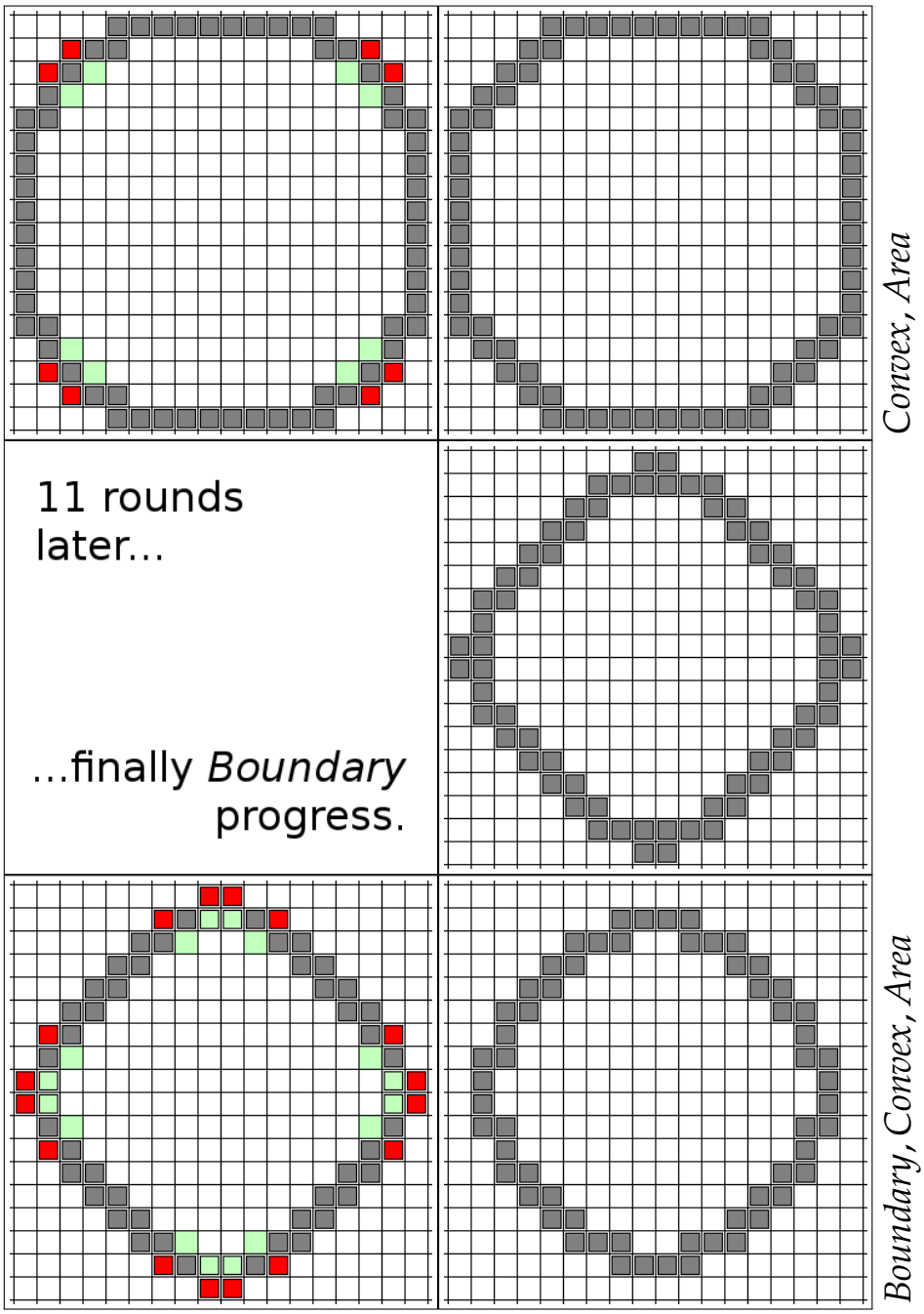



Area
Convex, Area
Area








11 rounds
later...
...finally Boundary
progress.
Convex, Area
Boundary, Convex, Area

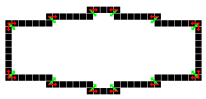
CHAPTER 5

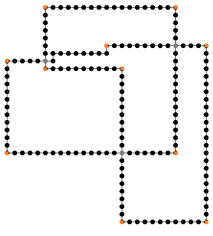
Asymptotically Optimal Gathering on a Grid

GRID-GATHER Strategy

Extended Basic&Plain Robot Model

In this chapter, we deliver a strategy that significantly improves the $\mathcal{O}(n^2)$ upper running time bound of OBLIVIOUS-GATHER (Chapter 4) by a factor of n. For this, we gently extend the robot model by a constant-sized memory and states that are visible to local neighbors to the *Extended Basic&Plain* robot model. We solve the local gathering problem of a swarm of n indistinguishable, point-shaped robots on a two-dimensional grid in asymptotically optimal time $\mathcal{O}(n)$ in the fully synchronous $\mathcal{FSYNC}$ time model. Given an arbitrarily distributed (yet connected) swarm of robots, the gathering problem on the grid is to locate all robots within a 2×2-sized area that is not known beforehand. Two robots are connected if they are vertical or horizontal neighbors on the grid. The locality constraint means that no global control, no compass, no global communication and only local vision is available; hence, a robot can only see its grid neighbors up to a constant L_1-distance, which also limits its movements. A robot can move to one of its eight neighboring grid cells and if two or more robots move to the same location they are *merged* to be only one robot. The locality constraint is the significant challenging issue here, since robot movements must not harm





the (only globally checkable) swarm connectivity. For solving the gathering problem, we provide a synchronous algorithm—executed by every robot—which ensures that robots merge without breaking the swarm connectivity. In our model, robots can obtain a special state, which marks such a robot to be performing specific connectivity preserving movements in order to allow later merge operations of the swarm.

5.1 Introduction

When developing the Oblivious-Gather strategy (Chapter 4), our goal was to supply a strategy that uses the same time model ($\mathcal{FSYNC}$) and the same, very minimalistic *Basic&Plain* robot model on the grid as Degener et al. [Deg+11] used for Go-To-The-Center in the Euclidean plane. Our runtime bound $\mathcal{O}(|\text{outer boundary}|^2) \subseteq \mathcal{O}(n^2)$, where |outer boundary| denotes the length of the swarm's outer boundary, matches the one of Go-To-The-Center [Deg+11].

In the current chapter, we want to improve the total running time ($\mathcal{FSYNC}$ time model).

Degener et al. [Deg+11] have proven that in the worst case their Euclidean Go-To-The-Center algorithm cannot perform faster than $\Omega(n^2)$. In Section 4.7, our experiments confirm that $\Omega(|\text{outer boundary}|^2)$ respectively $\Omega(n^2)$ seems also tight for our Oblivious-Gather grid strategy. So a different gathering strategy and an extended robot model seem to be required in order to improve the total running time. In this chapter, we develop such a strategy that gathers in time $\mathcal{O}(n)$ called Grid-Gather.

Our intention for speeding up the gathering is to focus on the fast shortening of the swarm's outer boundary. In order to be fast, much work (robot hops) must be done at the same time. In the $\mathcal{FSYNC}$ time model this cannot be done easily, because we have to take care that simultaneous robot hops do not break the swarm's connectivity. Introducing (visible) robot states allows many simultaneous hops without symmetry issues if, for example, only the robots with active state are allowed to execute a hop. For this, we extend the *Basic&Plain* robot model from Oblivious-Gather by a constant-sized memory and a constant number of locally visible states to the already introduced *Extended Basic&Plain* robot model. Figure 2.2 (page 15) shows an example.

Similar to the Oblivious-Gather strategy, two robots are connected if they

are located at vertical or horizontal (but not diagonal) neighboring grid cells. And vision is also restricted by a fixed constant L_1-distance on the grid.

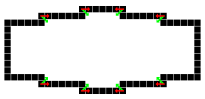
For our strategy, we adopt the idea of movable robot states from the Manhattan Hopper strategy from Kutylowski et al. [KM09] that has been invented for solving a completely different problem: shortening an arbitrary winding open chain between two fixed (unmovable) and distinguishable endpoint robots. In this strategy, the chain is built from statically connected robots, while chain neighbors must be located at L_1-distance 1. The strategy uses movable, so-called *run* states that can be generated by one of the endpoint robots. A run moves step-wise along the chain and allows only the robot that currently owns the run to perform a move/hop action. The authors show that for every started run, several steps later one of the robots that receives the run can perform a progress in the shortening of the chain. They further show that this even holds if multiple runs are active simultaneously. This enables a very fast $\mathcal{O}(n)$ total running time.

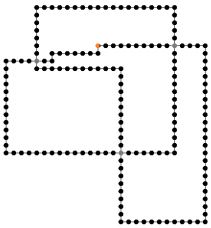
As mentioned, we want to do the gathering by fast shortening of the swarm's outer boundary. If we want to adopt the principle of movable run states, the outer boundary can be interpreted as a closed robot chain.

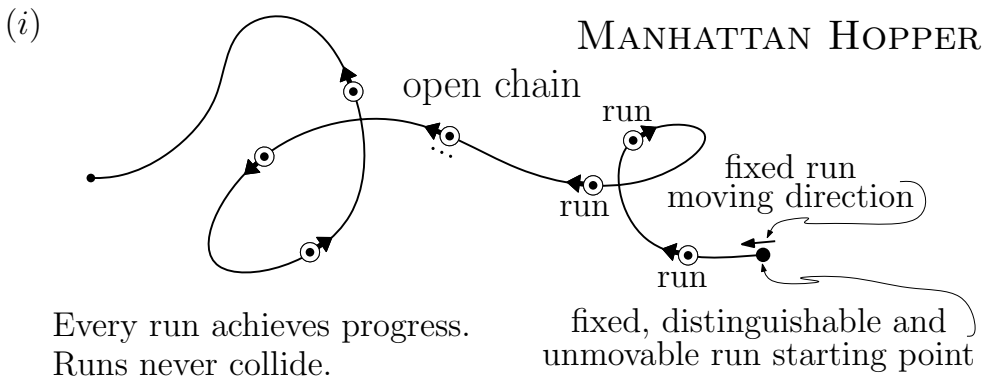
Figure 5.1 shows swarms for both strategies that are represented as Euclidean curves. White circles mark robots with active run state including an arrow that shows the run's moving direction.

We have to tackle several challenges:

#Boundaries:	In general, more than one boundary exist. Inner (dashed) and outer boundary cannot be distinguished
Run starting point:	Positions of run starting points must be determined by some local information (e.g., occupied cell pattern in the viewing range.). As robots move, run starting point positions change over time. Zero, one, or more run starting points can exist at the same time.
Run moving direction:	Moving directions must be determined individually, depending on local information. As a consequence, runs move in different directions.







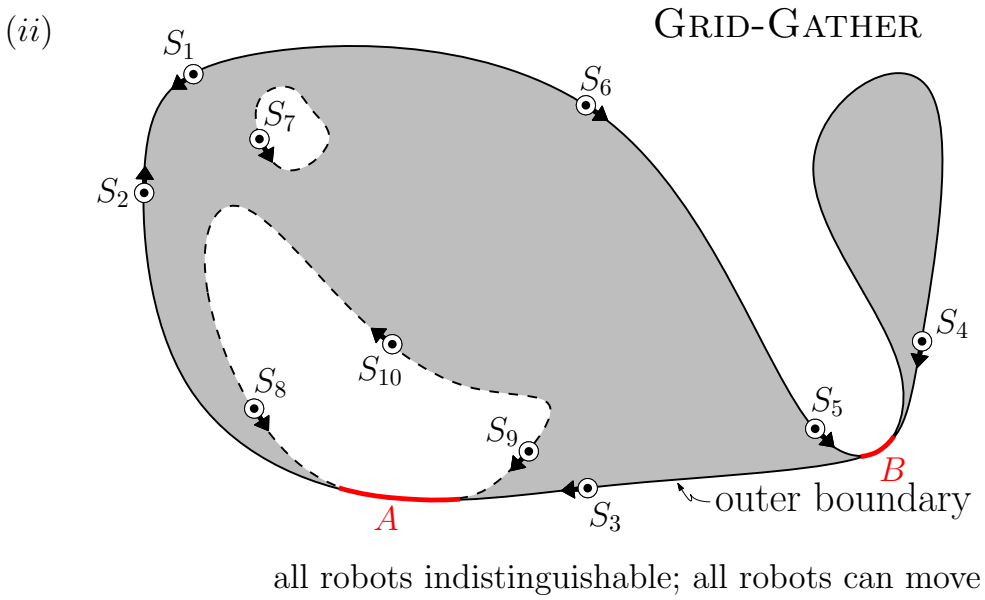
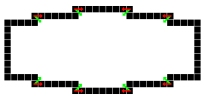


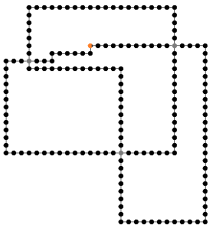
Figure 5.1: Moveable run states. (*i*): Manhattan Hopper strategy; (*ii*): Grid-Gather strategy

Boundary intersections:	Overlapping robots merge (See boundary parts A, B in Figure 5.1.(ii).). A robot can be part of different boundaries (A) or different parts of the same boundary (B) at the same time.
Collisions:	Collisions are possible because of different moving directions and overlappings: i.e., runs from the same (S_1, S_2; S_8, S_9; S_4, S_5) or different boundaries (S_3, S_8; S_3, S_9) can collide/intersect.
Gathering progress:	Many started runs never achieve progress.

In the following, after we have given a precise model definition, we construct closed chain substitutes from the swarm's boundaries and develop an $\mathcal{O}(n)$ gathering algorithm for the *Extended Basic&Plain* robot model.

Our local grid model. Our mobile robots need only simple capabilities: A robot moves on a two-dimensional grid and can change its position to one of its eight horizontal, vertical or diagonal neighboring grid points. A robot can see other robots only within a constant distance that we call *viewing radius* (measured in L_1-distance). We call the range of visible robots *viewing range*. The robots controlled by our algorithm need a viewing radius of L_1-distance 19. The robots have no compass, no global control, no IDs and no communication. A robot has a fixed small amount of memory to store a constant number of locally visible states. It can see the states of all robots inside the viewing range. Note that though for sake of simplicity we say a robot r that currently owns a run state "moves its state/run to a neighboring robot r'", this is implemented the other way round: When r owns the run, then r' can calculate on its own that it will receive the run next, because it can see the active run state (including its moving direction) of r. So, r' sets its run state on its own without requiring further information from r. r just unsets its run state in the next round.





Our algorithm uses the fully synchronous time model $\mathcal{FSYNC}$. Time is subdivided into equally sized rounds of constant lengths. In every round all robots simultaneously execute their operations in the common *look-compute-move* model [CP04], which divides one operation into three steps. In the *look step* the robot gets a snapshot of the current scenario from its own perspective, restricted to its constant-sized viewing range. During the *compute step*, the

robot computes its action, and eventually performs it in the *move step*.

Outline of the algorithm. As announced in the introduction, we start with developing closed chain substitutes for the given swarms in our *Basic&Plain* robot, connectivity and vision model. On the basis of these achievements we then develop the fast $\mathcal{O}(n)$ Grid-Gather gathering strategy that benefits from the concept of movable *run* states introduced in [KM09] (Manhattan Hopper strategy).

In the closed-chain based connectivity and vision model, each robot of a chain has exactly two well-defined neighbors given as part of the input of the problem.

Our substitute of this chain connectivity is the following: We call two robots *connected* if they are horizontal or vertical (but not diagonal) neighbors on the grid. This means that a robot is connected to at least one and at most four robots. Initially, under the restriction that the swarm is connected, the robots are arbitrarily distributed on the grid. A swarm is connected if the robots cannot be separated into two subsets such that no robot of the one subset is connected to any robot of the other subset and vice versa.

In order to find a substitute for the chain we define the so-called boundaries: The boundaries consist of all robots that have at least one empty neighboring grid cell. The black robots of Figure 5.2.(i) are part of the *outer boundary* and the hatched robots are part of *inner boundaries*. By that definition the swarm consists of one single outer boundary and typically multiple inner boundaries. A robot can detect if it is located on some boundary of the swarm, but because of its limited viewing range it does not know if it is located on the outer boundary or on some inner boundary.

The goal of our algorithm is to shorten the outer boundary closed chain substitute. However, the outer boundary does not have the same structure as the chain because the outer boundary can have fringes or even be shaped like an hourglass or overlap with inner boundaries, as the example in Figure 5.2.(ii) shows. Furthermore, because the robots can not distinguish between inner boundaries and the outer boundary, our algorithm will also shorten the inner boundaries.

Thus, our algorithm has to tackle two problems: First, we have to ensure that the algorithm will actually shorten the outer boundary. Second, we have to

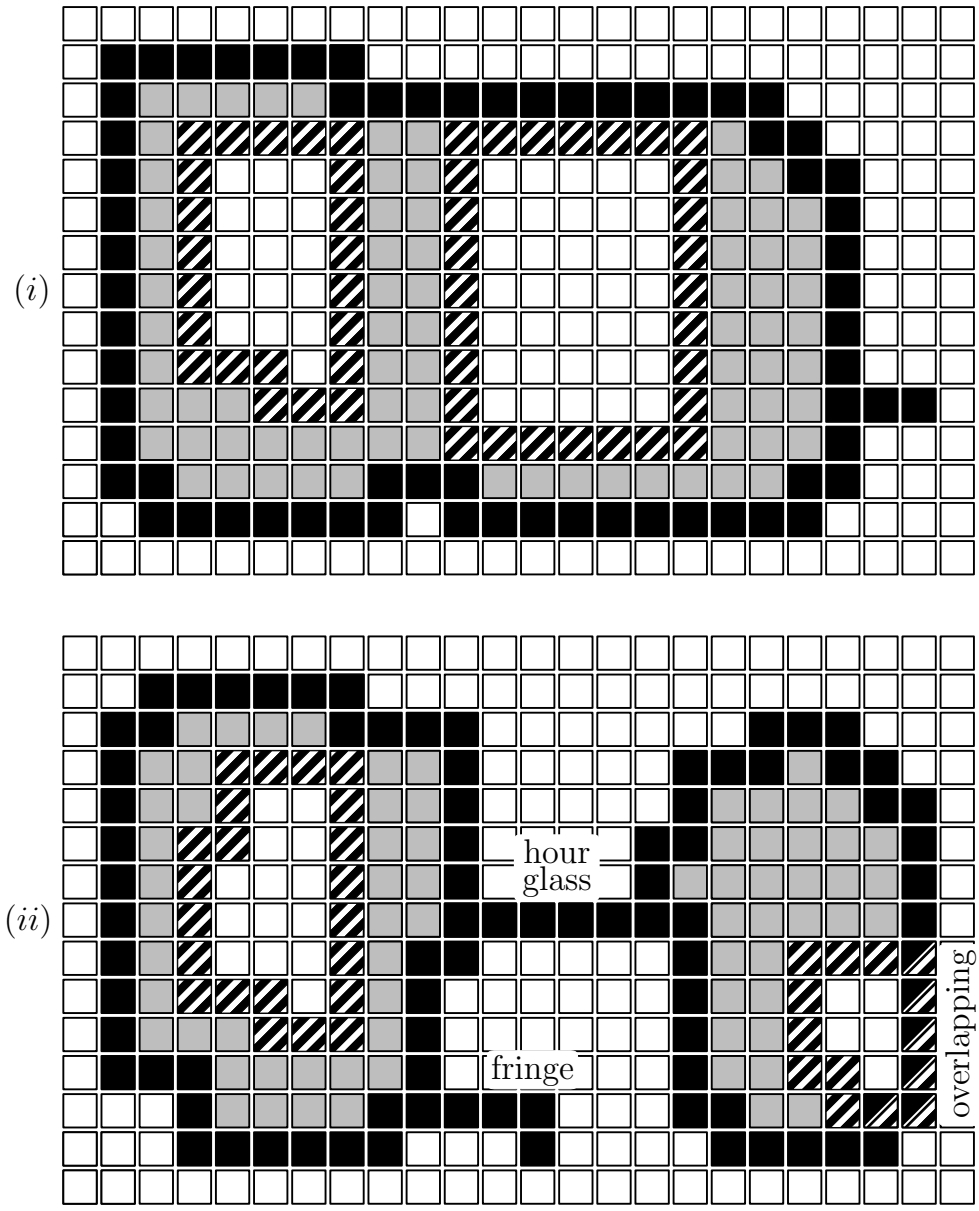


Figure 5.2: (*i*): Boundary definition. The hatched robots are part of *inner boundaries* and the black robots are part of the *outer boundary*. (*ii*): An hourglass-shaped swarm with an overlapping and a fringe.

ensure that the shortening of the inner boundaries does not disturb the shortening of the outer boundary. This is because runs from different boundaries can also intersect (cf. Figure 5.1.(*ii*) (see boundary parts A and B)).

The main idea for our algorithm is the following: Our algorithm achieves progress in the gathering by performing *merge operations*. A merge operation is a *hop* of a robot located on the boundary onto the same grid cell as one of its neighbors and the removal of one of them. A merge operation is only allowed if it does not disconnect the swarm.

Sometimes no merge is possible without disconnecting the swarm. In that case we want the robots of the boundary to reshape the swarm. Reshapement means that the robots reach positions such that new merges become possible, which does not break the connectivity. Reshapement operations consist of two types: Starting a reshapement and continuing a reshapement.

Starting a reshapement is allowed only to a robot that is located on a boundary and can see a certain configuration of robot positions in its viewing range. The robots that are allowed to start a reshapement generate a so-called *run state*, hop to another position, and give the run state to neighbors on the boundary. Robots receiving the run state are called *runners* and are allowed to continue the reshapement by hopping in the following round and giving/moving the state to neighbors of the boundary. In that way the run state will be moved along the boundary by the runners. If two certain runs converge until the corresponding runners can see each other inside their viewing range, then a merge can be performed. Gathering is finished when all robots are located within a 2×2 square, since in our model that situation cannot be simplified anymore.

For counting the progress of our algorithm, it is sufficient to consider the merges of the outer boundary only, i.e., we look at the shortening of the outer boundary. We will show that after a constant number of rounds either a merge is possible or new run states will be generated. The latter will provide at least two runs that converge and enable a merge after $\mathcal{O}(n)$ rounds. The runs move in parallel and our strategy ensures that the runners do not disturb each other. Different started run states result in different merges. Therefore the algorithm gathers a swarm of n robots in time $\mathcal{O}(n)$. Our result is asymptotically optimal for worst case swarms.

5.2 The Algorithm

In our model, merges/removals of robots mean progress of the gathering. Then, at the latest after $n - 1$ removals the gathering is done. In our model, a robot can be removed if two robots are located at the same grid cell. Then, one of them is removed.

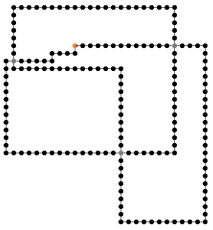
Because a robot's viewing range is restricted to just a constant size, these removals cannot be performed easily in general. Our algorithm performs two basic operations. More precisely, we have to deal with two cases:

1. Some neighboring robots perform a single hop such that afterwards at least two robots are located at the same position, while preserving the swarm connectivity. Then, we remove one of them. This is a so-called *merge* and further discussed in Section 5.2.1.

2. If on some parts of the swarm merges are impossible, we perform so-called reshapements to reshape the swarm in order to prepare merges in succeeding steps. These are explained in Section 5.2.2.

In Section 5.2.1, we now start with case 1. For the sake of simpler descriptions, we use terms like *horizontal, vertical, downwards, left,…* Since our robots do not have a common sense for this, the descriptions/figures are also to be understood in a mirrored or rotated manner.

5.2.1 Merges



A *merge operation*, or simpler a *merge*, is the simultaneous local operation of a sequence of neighboring robots that merges at least one robot, while not harming the overall connectivity of the swarm. There are different merge operations, all parametrized by a parameter k, as depicted in Figure 5.3. In the figure, the case $k = 1$ denotes the simplest variant, where only a single robot hops onto a grid cell occupied by another robot. Because the robot's viewing radius is > 1, namely 19, also bigger merges are possible within the local vision. Then, for not breaking the swarm's connectivity, multiple robots (the black ones in the figure) have to hop simultaneously: Using the local information about the occupied and empty cells in its viewing range, every robot can decide individually, on the basis of its local knowledge, if it participates in the

simultaneous merge operation or not. Specifically, the marked white grid cells must not contain any robot, any further marked cell must contain a robot, and the maximal length k of a merge configuration is limited by the viewing radius. Moreover, all not explicitly depicted cells are ignored for the decision.

The k robots that constitute a merge form a subboundary of the swarm. When operating, these subboundary robots simultaneously hop one grid cell in the same direction (in the illustration, this means downwards). The swarm's connectivity is ensured, since a merge is performed only if the depicted white cells are empty. After the hops of the black subboundary robots at least one cell contains two robots and hence at least one robot is removed, because in the merge operation definition gray robots do not move.

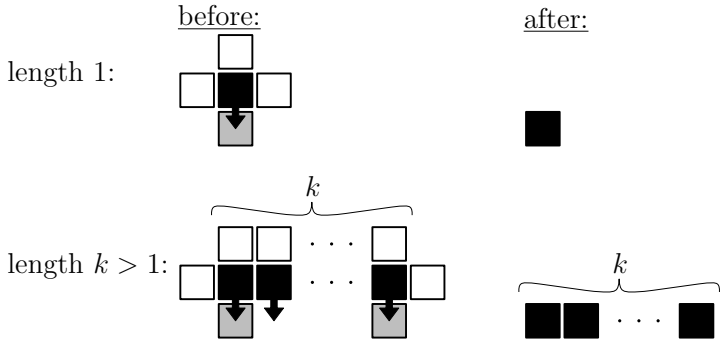


Figure 5.3: Robot subboundaries that allow progress hops (*merge operations*). The value of the *length k* indicates the number of robots (the black ones) of the subboundary. k is upper bounded by a robot's constant viewing radius. The operations are performed only if the cells, marked by white squares, are empty. Else, the swarm's connectivity might break.

Possibly, several simultaneous merges can occur at different parts of the swarm. The problematic cases we have to discuss are those where two of the robot subsets participating in the merge operations, i.e., the subboundaries, consisting of the black and gray robots of Figure 5.3, overlap (cf. Figure 5.4). Precisely, two cases need a closer look. We refer to the black and gray robots of Figure 5.3.:

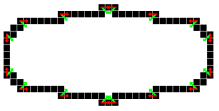
1. The beginnings and endings of the subboundaries overlap by two robots.
2. The beginnings and endings of the subboundaries overlap by three robots.

An example for 1. can be seen in Figure 5.4.*a*): (i): The robots of the subboundaries $1, \ldots, 6$ will all perform the hop as the black ones during the merge operation (Figure 5.3). In order to ensure a merge in this special case, not all of these subboundaries hop simultaneously: Every round, our algorithm executes the Merge step two times: at first only for length 1 and afterwards for all lengths > 1. Then, at first only the robots of the subboundaries 1 and 6 execute their merge hops. Here, two robots can be removed. Afterwards, the robots of subboundaries $2, \ldots, 5$ hop ((ii)): In the example of (ii), the special case occurs that after the hop, the robot a is not located at the same position as b and vice versa. So, no robot can be removed here. This does not matter, as the robots on the target cells of the outermost hopping robots do not move, so that there robots can be removed after the hop.

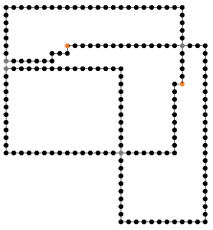
An example for 2. is shown in Figure 5.4.*b*). Here, the robot r belongs to the both subboundaries 1 and 2: i.e., concerning subboundary 1, r would hop downwards, but concerning subboundary 2 it would hop to the left. In this case, r performs a diagonal hop to the lower left, while the other robots perform their usual hops. Afterwards, r, a, b occupy the same grid cell and a, b are removed without breaking the connectivity.

5.2.2 Reshapement of the swarm by runners

From the global perspective, if nowhere in the swarm the algorithm's local merge operation is possible, then we call it a *Mergeless Swarm*. In this case, our goal is to perform certain *reshapements* of the swarm's outer boundary in order to make merge operations possible. Still considering the swarm from a global perspective, if there is a subboundary as depicted in Figure 5.5 by the black robots, then we continuously let an outermost robot of this subboundary perform the depicted diagonal hops. Eventually, this will shorten the subboundary enough to allow a (local) merge operation (cf. Figure 5.3). We say these hops *reshape* the subboundary.



For an individual robot with its limited viewing range this raises some challenges.



1. When a robot decides to start the reshapement, within its restricted viewing range it does not know if its task of performing diagonal hops will lead to merges (In Figure 5.5, this is the left outermost black one in

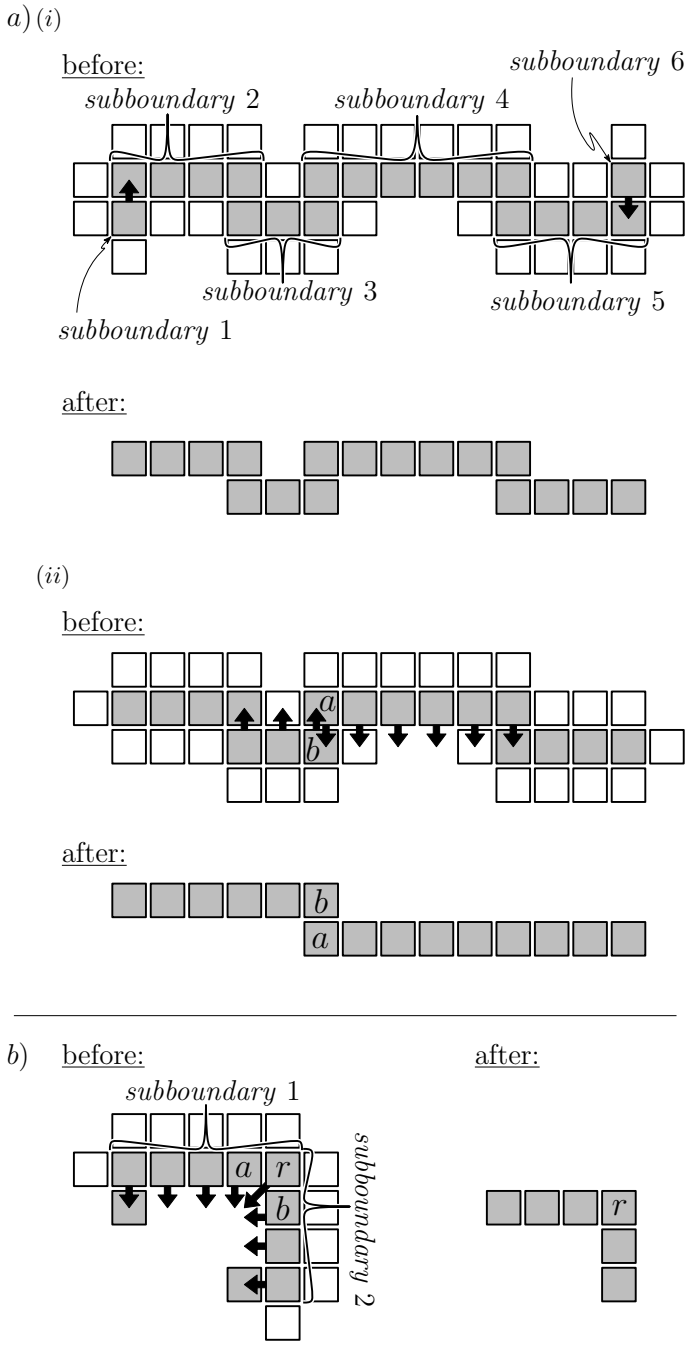


Figure 5.4: Overlapping merges. (significant examples)

round i.).

2. In the following rounds, due to the local viewing ranges and skewed coordinate systems, the local view of an outermost black robot may be the same as that of its gray neighbor. So, we have to ensure that the reshapement is continued by the outermost black robots instead of by their gray neighbors.

3. When robots decide to start the reshapement, symmetries of the subboundary's local shape may lead to breaking of the swarm's connectivity. In Figure 5.6 this is the case if both r and r' start the reshapement.

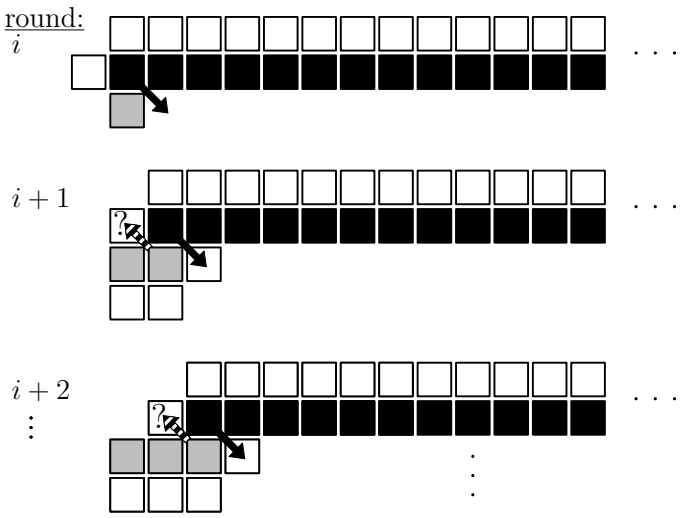


Figure 5.5: If the length of the black subboundary (i.e., the value of k in Figure 5.3) is larger than the robots' viewing radius, we shrink it by letting one or both of the outermost black robots perform diagonal hops. We indicate such hops by diagonal arrows.

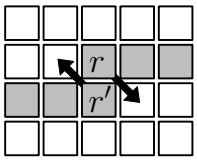


Figure 5.6: If r and r' both start reshaping the subboundary, the connectivity might break.

In the following, we tackle these challenges by introducing certain locally

visible states of a robot, which we call the *run* states. We call a robot with an active run state a *runner* and allow only runners to perform the reshapement hops. Robots can achieve the run state in two different ways:

start runstate: If the local subboundary within a robot's viewing range has a certain configuration, i.e., the relative positions of other robots and empty cells, then the robot decides on its own to generate the run state. We say such a robot starts a run. On the basis of the configuration of the local subboundary, the run state gets a fixed moving direction along the boundary. A robot can start and store up to two run states at the same time. Figure 5.8 shows how the local configurations must look like.

move runstate: A runner $R(S)$ can move the run state S to its boundary neighbor r' in moving direction of S. We say the run state has moved from $R(S)$ to r', while its in "start runstate" initially set moving direction always remains unchanged. Afterwards, r' is identified by $R(S)$.

Once a run state has been started in "start runstate", "move runstate" is executed in every of the following rounds. This means that the run moves along the boundary at constant speed and in the initially settled moving direction.

When starting runs, the shapes of Start-A and Start-B in Figure 5.8 ensure that the run starts cannot break the swarm's connectivity. For this, in a situation like in Figure 5.6 we do not start any runs. We name subboundaries consisting of shapes like in Figure 5.6 *quasi lines*. Figure 5.7 gives an example of a quasi line. Definition 5.1 contains the formal definition.

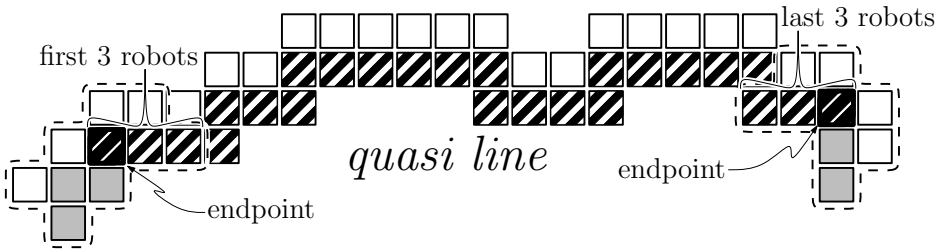


Figure 5.7: Example of a quasi line. The fat robots are its endpoints.

Definition 5.1 (quasi line)**.** We call a subboundary a horizontal *quasi line* if the following conditions hold:

1. At least its first and last three robots are horizontally aligned.
2. All its subboundaries of horizontally aligned robots contain at least three robots.
3. All its subboundaries of vertically aligned robots contain at most two robots.

In a Mergeless Swarm, at both ends of a quasi line a run starting subboundary Start-A or Start-B of Figure 5.8 in a matching rotation or reflection occurs. (If the swarm is not mergeless, then the subboundaries outside the quasi line's endpoints may also have other shapes than these.)

The definition of a vertical *quasi line* follows analogously.

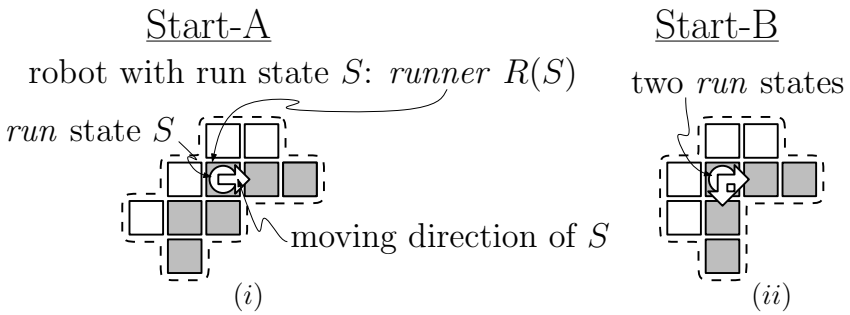
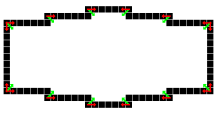
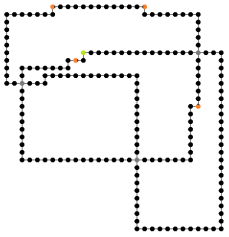


Figure 5.8: Run starting subboundaries: The robots marked by white circles decide to start the run states, only on the basis of the relative positions of the marked robots and empty cells. Here, gray squares denote robots, while white squares denote cells that must be empty. The white arrows indicate the moving direction of the runs. This is the notation we will use for marking a runner. $R(S)$ identifies the runner/robot which currently has the run state S; In (ii), Start-B, the robot marked by the circle is the endpoint of a horizontally and a vertically aligned subboundary at the same time. Here, we must start two runs moving in both directions along the boundary.



We let runs start at endpoints of quasi lines. We let them move along such quasi lines and, while doing this, perform reshapements of the boundary. For this, depending on the local shape of the subboundary, we require the run operations, shown in Figure 5.9.



a) OP-A: The runner and at least the next 3 robots are located on a straight line. Here, the runner first performs a diagonal hop, then moves the run to the next robot.

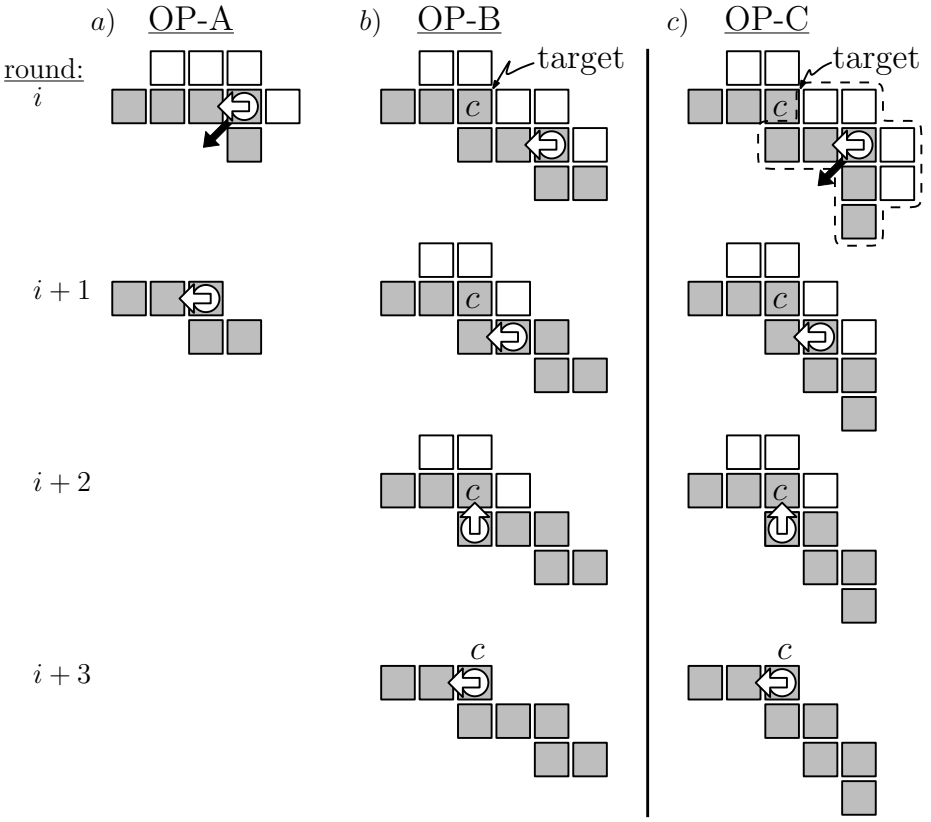


Figure 5.9: *a*) OP-A: Reshapement by a runner. The operation takes only one round. *b*) OP-B: No diagonal hops are performed until the target corner *c* is reached. *c*) OP-C: Special case when new runs start: First, perform one diagonal hop, then no diagonal hops until the target corner *c* is reached.

b) OP-B: The runner and only the next 2 robots are located on a straight line. Then, for 3 times the runners just move the run to the next robot without any diagonal hops. Afterwards, it is located at the target corner c.

c) OP-C: This one is needed at most once for a new run, if started at the run starting subboundary Start-B (Figure 5.8.(*ii*)).

These operations shall finally enable merges.

Two properties let multiple runs be active in parallel.

1. Usually, at different positions, the whole swarm contains multiple run starting subboundaries (cf. Figure 5.8) at the same point in time.
2. No matter if previously started runs are still active, every constant number of $L = 23$ rounds all robots simultaneously check if they can start new runs (cf. Figure 5.8) and if so, they do so.

Therefore, we have to deal with the event when two runs meet on a quasi line. We distinguish two cases (cf. Figure 5.10). *a*) This run pair has enabled the

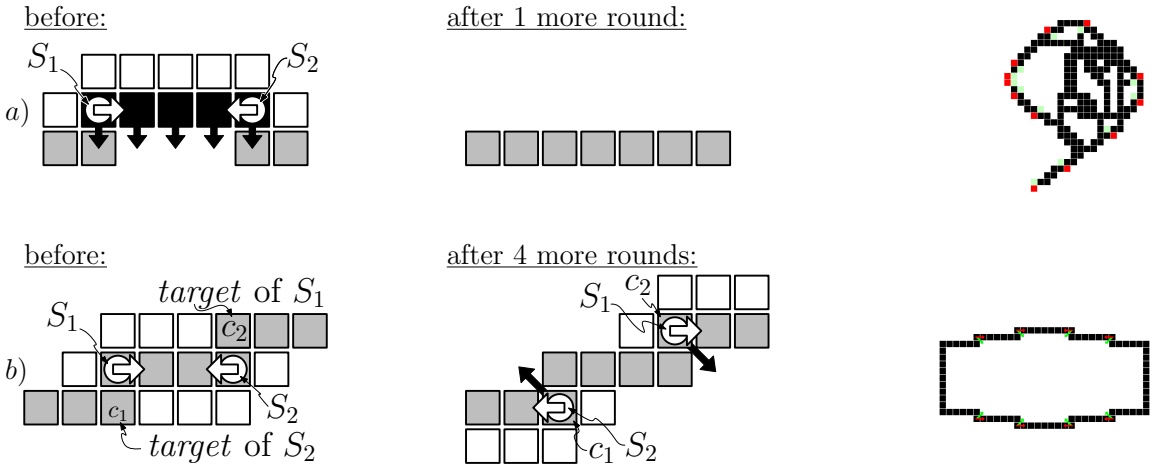


Figure 5.10: *a*) *Merge* operation: The reshapements of the runs S_1, S_2 have enabled a merge. This merge is performed and both are stopped. The operation takes only one round. *b*) *Run passing* operation: The runs S_1, S_2 cannot enable a merge. If their distance along the boundary is less or equal 3, then they pass each other by only keeping moving but without making the runners perform diagonal hops. Afterwards, i.e., when S_1, S_2 have reached their target robots/corners c_2, c_1, they return to normal operation.

desired merge. *b*) This pair is oriented in a way that does not enable a merge.

We let the runs of such pairs pass along each other. Figure 5.10.*b*) shows how this is performed for the case that both runs are located on the same quasi line: At the time when their distance along the boundary (i.e., the number of robots on the subboundary connecting both +1) is 3 or less, they only keep moving along the boundary, but the runners do not perform reshapement hops. We call 3 the *run passing distance*. This is repeated until S_1 is located at its target robot c_2 (then also S_2 is located at its target c_1). We call this the *run passing* operation. Afterwards, the normal reshapement operations are continued. In Section 5.5, we provide a full explanation of all possible situations of needed run passing operations. While in the easy case a viewing radius of 11 is the minimum value that suffices, we then need the value of 19.

5.2.3 Stopping runs

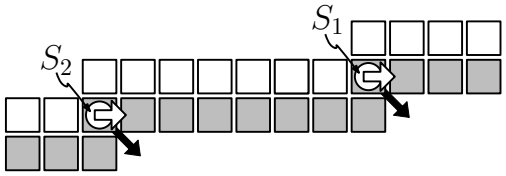


Figure 5.11: The *distance* between S_1, S_2 is 8.

We use the example of Figure 5.11 for introducing some terms when speaking about runs: Because S_1, S_2 are moving in the same direction, we call them *sequent* runs, while relative to their moving direction S_1 is located *in front of* S_2. The *distance* between them is defined as the number of robots on the subboundary connecting both +1. We say S_1 is *visible* to S_2 if the *distance* between them is ≤ 19.

In order to ensure that runs, that are active at the same time can work correctly, some of them have to stop in certain situations. More precisely, we let a runner $R(S)$ stop/terminate its run S if one of the conditions of Table 5.1 is true.

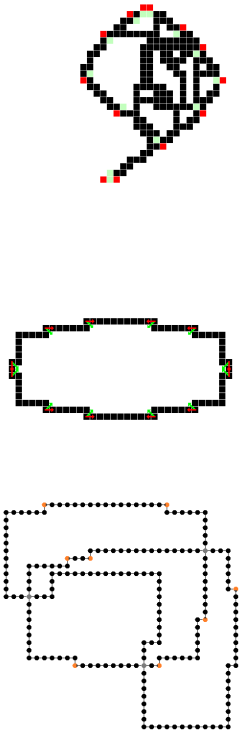
We give some more detailed explanations concerning some of the conditions of Table 5.1:

2) For example, because of merges two sequent runs may come too close to each other, which then might hinder the reshapements and the pipelining

A runner stops/terminates its run if at least one of the following conditions is true:

1. It was part of a merge operation.
2. It can see the next sequent run in front of it (This happens if sequent runs have come too close to each other, e.g., because of merge operations.).
3. It can see the quasi line's endpoint in front of it.
4. While it performs the run passing operation (Figure 5.10.*b*)), some change of the subboundary's shape prevents the operation from being completed successfully. (This can happen because of a merge.)
5. While it performs the operation OP-B or OP-C (Figure 5.9), some change of the subboundary's shape prevents the operation from being completed successfully. (This can happen because of a merge.)
6. While it performs the operation OP-A or OP-C, it has hopped onto an occupied cell. (Then, one of both robots is removed.)

Table 5.1: Conditions that let a run terminate.



(Section 5.3.2). The affected runners can detect this on their own. The criterion for this is that the next sequent run in front of them becomes visible. Then, the termination condition 2) matches and the run behind stops.

4) We assume that in Figure 5.10.*b*) to the right of robot c_2 another run S_3, moving in the same direction as S_2, is located. During the rounds in which S_1 and S_2 are performing their run passing operation, S_3 keeps moving towards c_2. Now, for example, it may happen that because of the reshapements of S_3, c_2 becomes part of a merge operation. Then, c_2 would hop downwards such that the corner shape does not exist anymore. Because this corner has been the target of S_1, S_1 could not continue its reshapements after the run passing, so it terminates.

In summary, all robots synchronously execute the algorithm, shown in Figure 5.12.

5.3 Why the Strategy Produces Progress in Gathering

For measuring progress, we only consider runs started pairwise at both ends of certain quasi lines. The kind of run pairs that in any case lead to a merge are called *good pairs*.

5.3.1 Good pairs

Assume a newly started run pair, connected by a quasi line. We call this pair a *good pair* if the following is true (cf. Figure 5.13): The exterior neighbors (the fat bordered robots in the figure) of the run pair are both located downwards and the whole (global) area above the connecting quasi line does not contain any robot (The same definition analogously holds for any rotation.). Looking at our run operations OP-A, OP-B and OP-C as well as the run passing operation and merges, we notice that none of them could let a robot which is not part of the considered quasi line hop into this empty area.

Such good pairs always enable a merge if they have been moving close enough together and then terminate. Figure 5.14 shows an example of a good pair where the shape of the quasi line connecting both is a straight line. The

Every robot r in every round checks the following three steps:

1. Merge: (In each round executed two times: First execution: Robots perform merges only if length $= 1$. Second execution: The robots perform merges only if length > 1.)
   If r detects a possible merge within its viewing range then
   - if r is one of the black robots in Figure 5.3, it hops downwards.
   - if afterwards r is located at the same position as one of the gray robots, the gray one is removed without breaking the connectivity.
2. Run Operations: If r is a *runner*, then
   a) Its run terminates/stops if any of the conditions of Table 5.1 is true.
   b) Runner's Movement and Reshapement
      - Run passing:
        - If r is currently in progress of executing the run passing operation (Figure 5.10.*b*)), then this operation is continued.
        - Else, if r can see a run in front of it, such that both are moving towards each other and the distance between them is less or equal than the *run passing distance*, then r starts the run passing op.
      - If r is not in progress of passing, then
        - If r is in progress of executing a run operation OP-B or OP-C (cf. Figure 5.9), which take more than one round, then this one is continued.
        - Else: r executes the matching new run operation OP-A, OP-B or OP-C.
3. Start new runs: Every $(L = 23)$th round, r checks if it can start a new run:
   If r is one of the robots marked by circles of the run starting subboundaries Start-A or Start-B (cf. Figure 5.8), then it starts one resp. two runs.

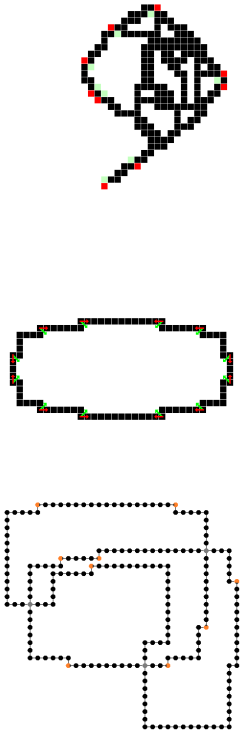


Figure 5.12: The algorithm.

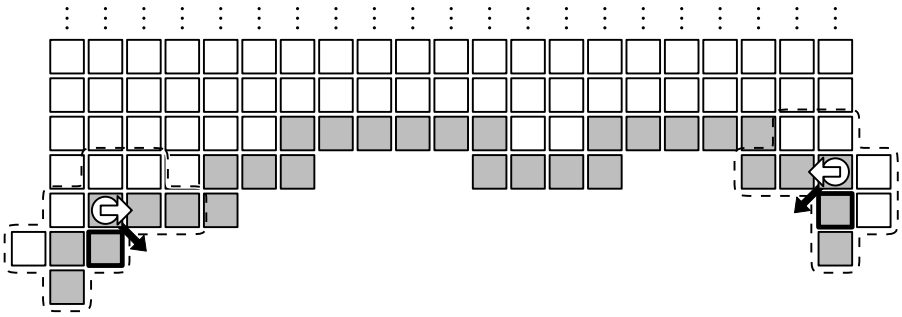


Figure 5.13: Pair of runs, connected by a quasi line. The runs are a *good pair* if the outer subboundary neighbors (the fat robots) are both located downwards and globally the whole area above does not contain any robots.

example of the figure shows that using runs, we solve the problem we have had in Figure 5.5, where it was impossible to locally decide by the robots which of them has to perform the reshapement hop. Figure 5.15 shows an example for a more general case of a quasi line: Again, the black subboundary will be reshaped for performing a merge. But now, several run operations (cf. Figure 5.9) are needed until a runner arrives at an endpoint of the black subboundary: Until round $i + 4$, the runners execute only the operation OP-A. Afterwards, the runner $R(S_r)$ starts operation OP-B, while $R(S_\ell)$ still executes OP-A. After operation OP-B has been processed completely, S_r finally is located at the right end of the black subboundary (round $i + 8$). Now, its runners start shortening this subboundary by executing OP-A until the merge can be performed. Afterwards, S_ℓ is still active and keeps moving and will stop at the latest when an endpoint of the quasi line becomes visible.

Many of the started runs do not belong to good pairs. In Section 5.4, we will prove the actual existence of good pairs.

5.3.2 Pipelining

We will show that even if multiple good pairs are nested into each other, different good pairs will enable different merges. This is what we call *pipelining*. Figure 5.16 shows an example for the case that the quasi line has the shape of a straight line. The more general case works analogous to this.

As runs are moving with constant speed, the runs of the inner good pair will meet first and, as the result, first enable a merge and stop. Then, obviously

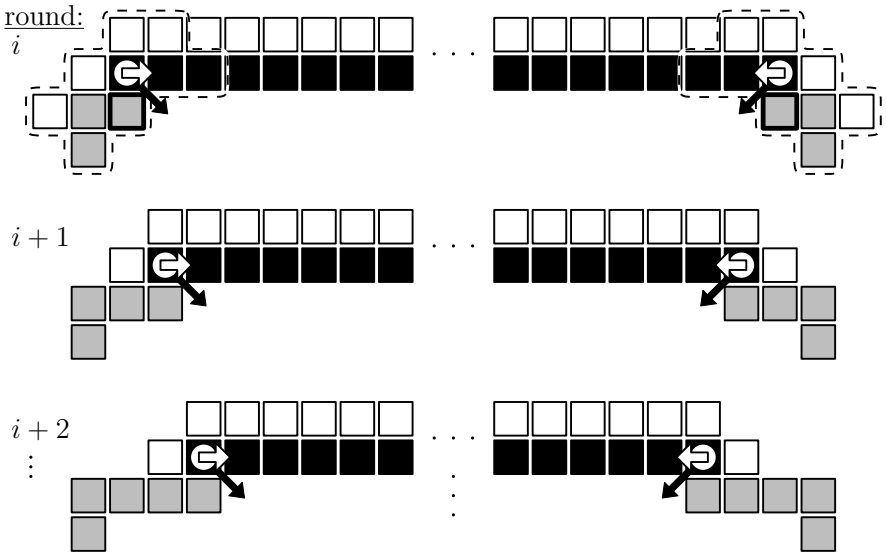
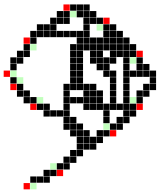


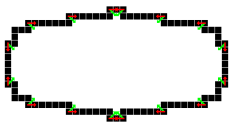
Figure 5.14: A *good pair* of runs. In round i, the new runs start (cf. Figure 5.8). Afterwards, the action OP-A of Figure 5.9 is repeatingly executed. The runs move closer and closer together. If the runs have moved close enough, a merge can be performed.

the outer good pair will also enable a (different) merge, some rounds later.

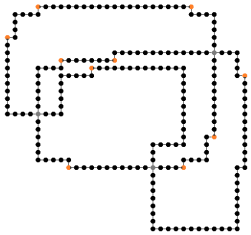


5.4 Correctness and Running Time

Our basic argumentation for the total running time is the following: If every round a merge can be performed, then the time needed for the gathering is obviously upper bounded by n, with n being the number of robots. If no merge can be performed, then every $L = 23$ rounds a *good pair* can be started which reshapes the swarm such that a merge can be performed after at most n further rounds. As L is a constant, this then leads to a linear total running time $\mathcal{O}(n)$.



In this section, we give the formal proofs. Our goal is the proof of the correctness and the linear running time (Theorem 5.4). In our main approach, we want to enable merges by good pairs if else no merge could be performed. The following two lemmas prove that this actually works. They are the base for the proof of the theorem. The proofs of the lemmas can be found in Sections 5.4.1 and 5.4.2.



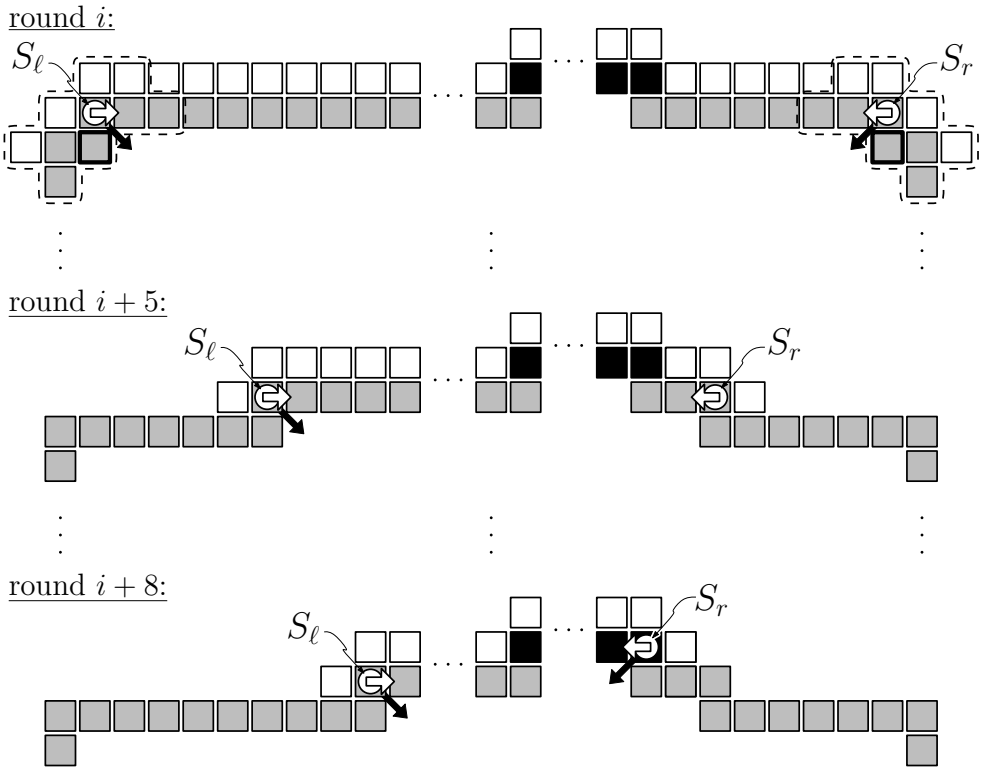


Figure 5.15: A good pair, started on a quasi line in order to shorten the black sub-boundary. Several and different run operations are needed.

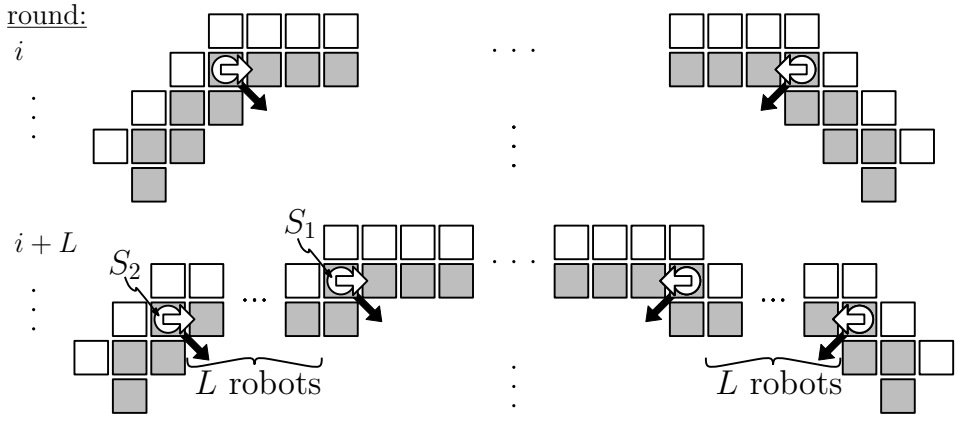


Figure 5.16: Pipelining of runs. New runs are started every $L = 23$ rounds.

Progress pairs. Because of the local vision, new runs and maybe also good pairs are started every $L = 23$ rounds, regardless of whether or not somewhere in the swarm merges can be performed. In the following analysis, we will argue only with new good pairs that are started if during the last $L - 1$ and the current round in the whole swarm no merge has been performed. We distinguish such good pairs from others by calling them *progress pairs*.

Lemma 5.2. *Every $L = 23$ rounds either a merge has been performed or else a new progress pair is started.*

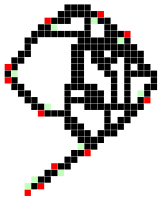
Lemma 5.3. *For all progress pairs the following properties hold.*

a) Every progress pair enables a merge (after at most n rounds).

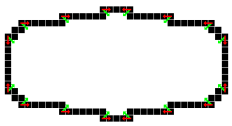
b) Different progress pairs enable different merges.

Using these two lemmas, we can now prove the total linear running time.

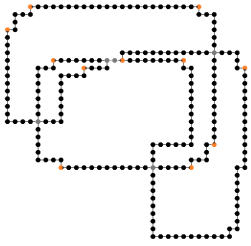
Theorem 5.4. *Given a swarm of n robots. Then, after $\mathcal{O}(n)$ rounds gathering is done. This is asymptotically optimal.*



Proof. We subdivide time into intervals of lengths L, where L denotes a constant number of rounds. Merges can be performed during at most n such intervals, because every merge removes at least one robot. In all other intervals a new progress pair starts (Lemma 5.2). Each of these progress pairs leads to a merge (Lemma 5.3.*a*)). Because no two of them lead to the same merge (Lemma 5.3.*b*)), the number of intervals without merges is also upper bounded by n.



By Lemma 5.3, a progress pair needs at most n rounds until it has led to a merge. We assume the worst case, which is that in the last of the $2 \cdot n$ intervals the last progress pair was started. Then the total running time is upper bounded by $2n \cdot L + n$, which proves the upper bound of the theorem, because L is a constant.



In our model, the diameter of the initial configuration provides the worst case lower bound $\Omega(n)$ for any gathering strategy. □

5.4.1 Proof of Lemma 5.2

Proof. The lemma requires that the swarm is a *Mergeless Swarm*. As the basis of our proof, we start by showing that then the outer boundary of the swarm only

consists of quasi lines and stairways (cf. Figure 5.17). First, we assume that every robot is connected to only two other robots. Then, especially no robots are located at the inside of the swarm, and the swarm consists only of its *outer boundary*. We start with the assumption that merges are possible only up to the length 2 (cf. Figure 5.3). Then, as by definition all horizontal subchains of a horizontal quasi line consist of at least 3 robots, no merge can be performed on them. In order to connect two horizontal or two vertical quasi lines without enabling a merge, they must be connected by so-called *stairways*. Stairways can have arbitrary length and are subchains of alternating left and right turns. Figure 5.17 shows an example. The bicolored robots are the connection points between the quasi lines and the stairway. All differently shaped connecting

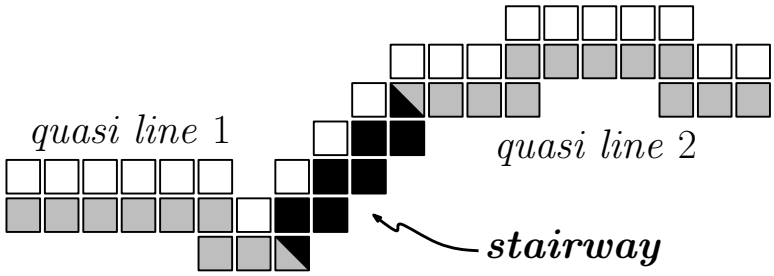


Figure 5.17: Two quasi lines, connected by a *stairway*.

subchains would allow merges. Stairways can also connect a horizontal and a vertical quasi line. Therefore, assuming that every robot is connected to only two other robots, the whole outer boundary consists only of quasi lines and stairways.

This is still correct if we extend the allowed merge operations to length > 2, because executing such a merge operation on a quasi line does not harm the quasi line property of the corresponding subchain (cf. Definition 5.1 (quasi line) and Figure 5.3 (Merge operation)). Hence, this only reduces the number of "allowed" quasi lines.

Now, we also allow connectivities bigger than 2, i.e., a robot can have more than 2 horizontal or vertical neighbors. Then robots may also be located at the swarm's inside or different parts of the outer boundary may be neighboring (i.e., if at that position the diameter $= 2$). Such robots can hinder merges. Figure 5.18 shows examples for configurations with existing inside robots. In the figure, the dashed bordered robots are part of the outer boundary. The

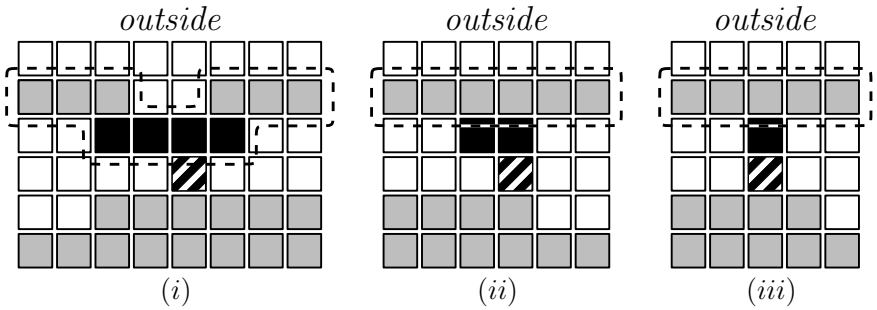
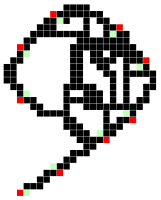
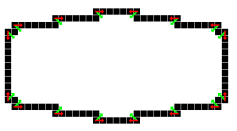


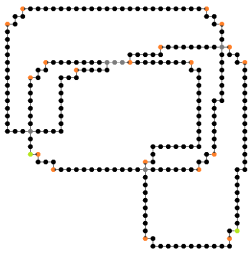
Figure 5.18: The hatched robot prevents the merge.

hatched robots prevent the black ones from performing a merge for which the black ones needed to hop upwards. So, although the swarm is mergeless, the outer boundary may still contain mergable shapes. For merges of length > 2 ((i) in the figure), the outer boundary then still is a quasi line. For merges of length 1 and 2 ((iii, ii) in the figure), the black robots are not part of the outer boundary. So, in every *Mergeless Swarm*, the outer boundary consists only of quasi lines and stairways.



Now, we can show that in a *Mergeless Swarm* a progress pair always exists. For this, we assume a global north and west and construct a vector chain along the swarm's outer boundary as follows. We consider the upper envelope of the swarm and take its left- and rightmost robots s respectively t. Because s, t must be located on the boundary of the swarm's smallest enclosing rectangle and furthermore the swarm is mergeless, below each of them two additional robots must exist (cf. Figure 5.19). We start from the left at the vector $\vec{v_0}$ constructing a vector chain in clockwise orientation along the outer boundary of the swarm, ending at the vector $\vec{v_m}$. The figure shows a significant example of this construction. Here, the gray colored robots denote the corresponding part of the swarm's outer boundary. Starting from its left end, we divide this vector chain into longest x-monotone subchains. The vector, originated at s, by construction points to the east. The second subchain starts when the first vector points to the west. The third subchain starts when the first vector points to the east again, and so forth. In the figure, these are the vectors $\vec{v_{16}}$ respectively $\vec{v_{26}}$. The constructed vector chain may overlap itself at places where the diameter of the swarm's boundary amounts only 1, but cannot contain any crossings. Because all constructed subchains are x-monotone, at least one of them is fully





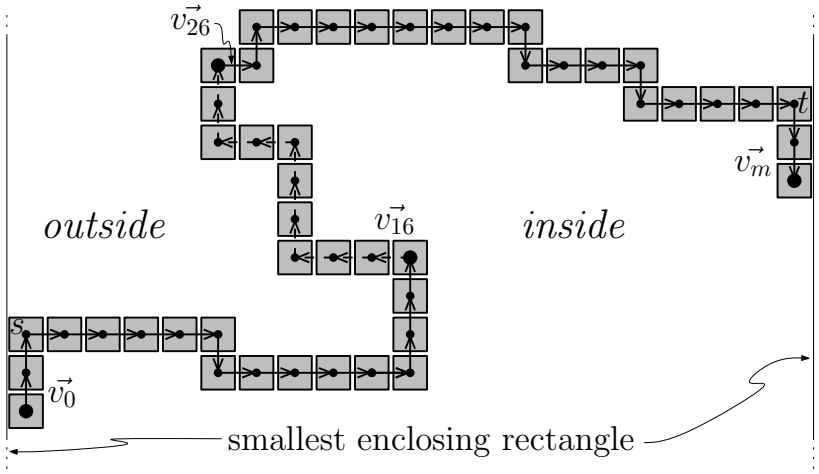


Figure 5.19: Constructing a vector chain along the swarm's outer boundary.

part of the upper envelope. Let $\mathcal{C}$ denote this subchain. If we consider $\mathcal{C}$ then, as the swarm is mergeless, both predecessors of its first vector must point to the north, i.e., the corresponding robots are located downwards. By the same arguments, both successors of its last vector must point to the south, i.e., the corresponding robots are also located downwards. This also holds if s or t are endpoints of this vector chain. Then, on both sides three neighboring robots are on a vertical straight line. By Definition 5.1 of a quasi line, these robots must be part of vertical quasi lines.

In contrast, if looking at the subchain $\mathcal{C}$, then, because the swarm is mergeless, $\mathcal{C}$ must contain at least two succeeding vectors, both pointing to the east. The three connected robots then must be part of a horizontal quasi line. At the transition between horizontal and vertical quasi lines, possibly connected by stairways, the shape matches the starting subboundaries Start-A or Start-B of Figure 5.8. As both vertical quasi lines are located downwards w.r.t. the horizontal quasi line of $\mathcal{C}$, a good pair can be started (cf. Section 5.3.1). Because we have assumed that the swarm has been a Mergeless Swarm also during the previous $L-1$ rounds, this good pair is a progress pair. □

5.4.2 Proof of Lemma 5.3

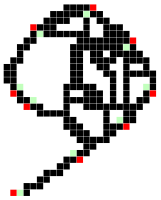
For the proof, we need some run invariants stated in Lemma 5.5.

Lemma 5.5. *The value $L = 23$ and the value 19 for the viewing radius ensure that for every run S until it terminates, the following invariants hold.*

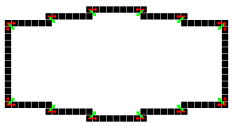
1. *Every round, S moves one robot further in moving direction.*
2. *After the first three rounds after the start of S, the reshapements of the runner $R(S)$ do not violate the quasi line Definition 5.1 of its own quasi line.*
3. *The reshapements of $R(S)$ do not violate the quasi line shape of other quasi lines that belong to good pairs.*
4. *S cannot see other sequent runs in front of it.*
5. *S is either in progress of passing along another run or the runner $R(S)$ executes one of the operations OP-A, OP-B or OP-C (Figure 5.9 on page 72).*
6. *Good pairs stay being good pairs.*

Now, we can prove Lemma 5.3:

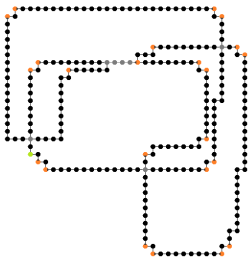
Proof. *a)* : At the time, a progress pair S, S' is started, the subboundary connecting both is a quasi line. Because of Lemma 5.5.2) and 3), this also does not change if other runs are located on this quasi line or next to it. And also merges preserve the quasi line properties. So, if not stopped, S and S' keep moving towards each other (Lemma 5.5.1)). Because of Lemma 5.5.6) a merge can be performed at the latest when they meet, which takes at most n rounds.



It remains to show that S, S' are not stopped by the algorithm's termination conditions (Table 5.1 on page 75) before the merge could be performed. In the following, we check all these conditions. In this proof, we denote the quasi line on which S, S' are located by the variable q.



2): Let $S^\star$ be the next sequent run in front of S. By definition of a progress pair, no merge has been performed since the last time new runs have been started. This means that if no run from outside the quasi line q has interfered with q, the distance between S and $S^\star$ is at least $L - 3$ (cf. the proof of Lemma 5.5.5)), which is bigger than the viewing radius. (The same analogously holds for S'.) So then, the runs of a progress pair cannot be stopped by this termination condition. So, we assume that other runs have interfered with q. If $S^\star$ originates from a different quasi line, S and $S^\star$ can only be oriented in the way shown in Figure 5.20, because the run operations require the cells



"above" to be empty. If these quasi lines shall meet, then at least one of them cannot be y-monotone. But then a merge would have removed at least one of them before both runs have come too close. So $S^\star$ cannot originate from a different quasi line.

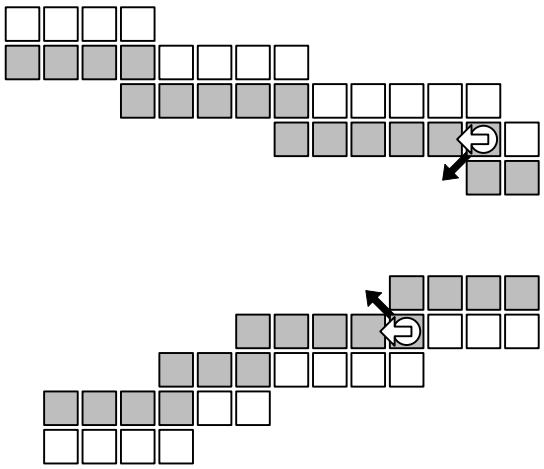


Figure 5.20: Too close sequent runs cannot originate from different quasi lines.

So, a run on a different quasi line (lying alongside q) must have shortened the part of q that connects S and $S^\star$. This can only have happened because of a merge operation, but this is not possible, because S, S' is a progress pair.

3): If a run of a progress pair can see an endpoint of the quasi line in front of it then, because of Lemma 5.5.2) and 3), the other run of the progress pair must have previously been stopped. As a run of a progress pair cannot be stopped by condition 2), it instead must have been stopped because of a merge. So, a termination because of condition 3) is allowed.

4, 5): The change of the quasi line's shape could either have happened because of the reshapement of a sequent run or because of a merge operation. Because we have ensured the minimum distance of two sequent runs to be large enough, the first case cannot have happened. So, a merge must have been the reason. Then, this merge must have been enabled by another run $S^\star$, moving towards S. If $S^\star$ is the partner run of S, then stopping is allowed. Else, because progress pairs are nested into each other, $S^\star$ either was not a run of a progress pair or its partner run has previously been stopped. In the latter case, because runs of progress pairs cannot be stopped because of condition 2), the

run must have been stopped by an earlier merge. In both cases, the current merge can be credited to the progress pair of S.

1): By the same arguments, this merge does not need to be also credited to a different progress pair. This then also proves b) of the lemma. □

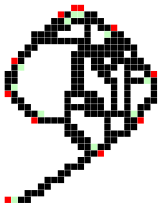
The missing proof of Lemma 5.5 can be found in Section 5.4.3.

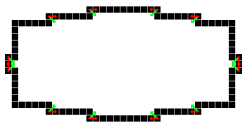
5.4.3 Proof of Lemma 5.5

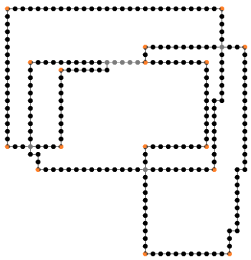
Proof. Let q be the quasi line S is located on and w.l.o.g. we assume that q is a horizontal quasi line. 1): This directly follows by the run definition in Section 5.2.2.

2): Cf. Figure 5.9 on page 72. OP-A ensures that on horizontal quasi lines only horizontal subchains of lengths > 2 are shortened and vertical subchains remain unextended. OP-B does not reshape. OP-C can be executed only during the first three rounds after S has been started. Our design of the run passing operation also ensures that the quasi line properties are maintained.

3): Because of the run stopping condition of Table 5.1.3) on page 75, $R(S)$ can only reshape a quasi line $q^\star$ if $q^\star$ is also a horizontal one. As the run operation OP-C cannot be applied inside $q^\star$, we only need to consider OP-A. If the cell onto that $R(S)$ hops is empty, then OP-A (because of the cells that are required to be empty) can be executed only if S is also located on $q^\star$, because of the good pair definition. But then, by the same arguments as in 2), the reshapements of $R(S)$ cannot violate the quasi line definition. If in contrast the cell onto which $R(S)$ hops is occupied by some of the robots of $q^\star$, then this operation does not modify the shape of $q^\star$.







4): This is ensured by the run termination condition of Table 5.1.2).

5): Cf. Figure 5.9. All run operations OP-A, OP-B and OP-C ensure that if S was located at some corner c_1 when the operation started, it afterwards either is located at some other corner c_2 such that both corners are rotated equally or terminates if the target corner has been removed during the operation (cf. Table 5.1.5, 4)). Then, because still located on a quasi line (cf. 2)), again OP-A or OP-B can be applied or run passing is started.

The run passing needs a closer look because it interrupts other operations. In order to ensure a regulated behavior, we chose the distance between sequent runs big enough such that a run does not have to execute a new run passing

operation before it has finished its previous one. As the value of the constant L controls the length of the waiting intervals between the start of two sequent runs and our stopping conditions maintain this minimum distance by stopping a run if the next sequent run in front of it becomes visible, we need to settle the values for these constants appropriately. At this place, we only analyze the case that was explained in Section 5.2.2, i.e., all participating runs are located on the same quasi line. The other cases (Section 5.5) can be proven similarly. We look at two sequent runs S_1 and S_1^{succ} such that S_1^{succ} has been started after S_1. Their distance D is at least $L - 3$. This value is achieved if at the start of S_1 the operation OP-C and next, OP-B was executed.

Now, we chose the value for the constant D big enough for ensuring that while some run S_2 is passing along S_1, S_1^{succ} becomes visible to S_2 the earliest when the passing operation with S_1 has been completed. Figure 5.21 shows an example for the longest possible duration of a run passing operation. Here,

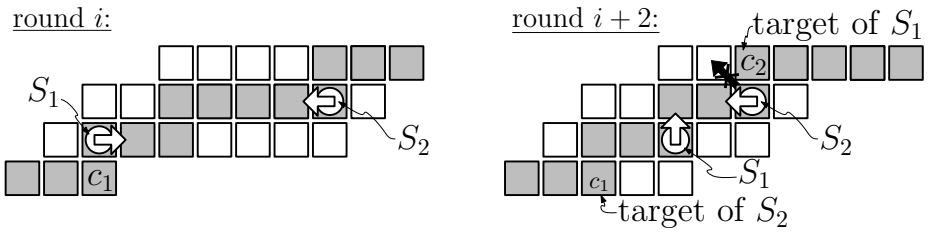


Figure 5.21: Runs, passing along each other while the runner $R(S_1)$ is executing the operation OP-B.

after the run passing has been started, it takes 6 rounds until S_2 has arrived at its target corner. Because the run passing operation starts when the distance between S_1 and S_2 is ≤ 3, after the passing operation, the distance between S_2 and S_1^{succ} equals $D - 9$. We want this then to still be ≥ 3 so that S_1^{succ} is not visible to S_2 before this point in time. So we choose $D \geq 12$ and together with the above argumentation concerning the minimum distance between sequent runs it follows $L \geq 15$. In order to detect that the distance has become smaller than 12 $(= D)$ in which case we have to stop a run (cf. Table 5.1.2)), the viewing radius must be 11.

Using the same construction, but including all other possible cases of run passings (Section 5.5), we can show that the values 7 for the *run passing distance*, $L = 23$ and a viewing radius of 19 suffice.

6): When defining good pairs in Section 5.2.2, good pairs have been characterized by the relative position of the outer direct neighbors of the good pair according to the quasi line. The first part of the proof of 5) finishes the proof. □

5.5 Run Passing Operation in Detail

If two runs S, S' that are moving towards each other, but do not enable a merge, have come too close to each other, then we let them pass along each other with a constant movement speed but without performing diagonal hops, in order to prevent the connectivity from breaking or destroying the quasi line property. We call this the *run passing* operation. If they have become close enough that the operation is needed, then also S can see S' within its constant viewing range and vice versa. For the run passing operation, it is important to assign a target corner to every such run so that it can continue its reshapements afterwards. Cf. Figure 5.22. The subfigures show two runs S, S' that need to

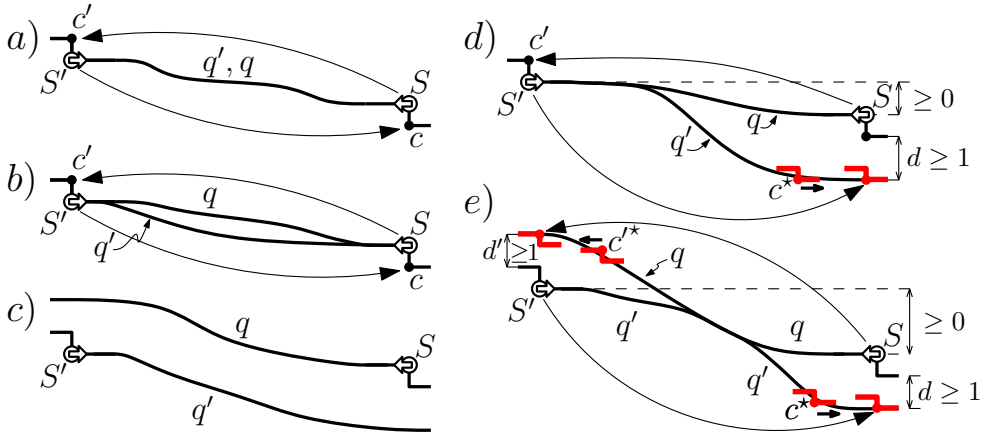
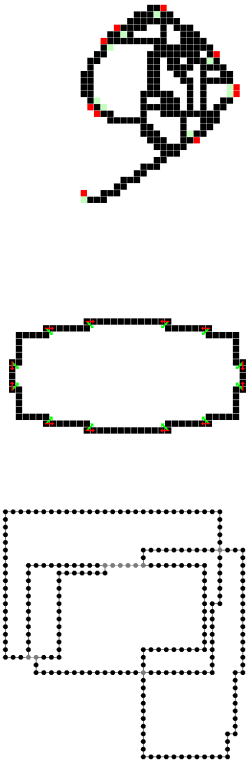


Figure 5.22: Classification of all possible cases that can occur if two runs need to pass along each other.

pass along each other. The curve q, originating at S symbolizes the quasi line that belongs to S. Analogously, the quasi line q' belongs to S'. We can assume that locally no robots are located in the area above q and below q', because else the corresponding run S respectively S' could not be part of a good pair so that it is allowed to stop itself. Then, the run passing is not needed anymore.



If a run passing operation is necessary then, between the locations of S, S', the quasi lines q, q' must in some way overlap. We now explain that in all possible cases of overlappings always suitable target corners can be assigned to both of the runs, so that they can continue their reshapements after the passing. We refer to Figure 5.22.

a) q, q' are identical. Then the run passing operation executes in the way explained in Section 5.2.2.

b) q, q' overlap at the location of S and at the location of S'. Then by the same arguments as in a), the target corners c, c' must exist.

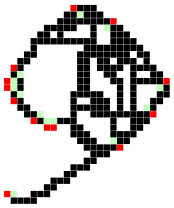
c) q, q' are disjoint. Then the following reshapements of S, S' will either make q and q' overlap or not. In the first case, the overlapping leads to the removal of robots so that S, S' are allowed to stop at this point in time. In the second case, S, S' can proceed their normal reshapement operations without interfering.

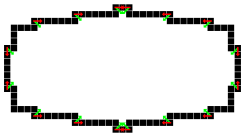
d) q, q' overlap only at the location of S'. Then, the vertical distance between S and S' must be ≥ 0 because else a merge would make a run passing superfluous. As on the right, q' is located below q, ($d \geq 1$), there must exist some corner $c^\star$ with the same rotation as the corner at that S' is located. The robots of q' then reconfigure this way that $c^\star$ is located at the right endpoint of the sub quasi line. Afterwards, S' can move to this corner and then continue its reshapements. In contrast, S can simply move to c'.

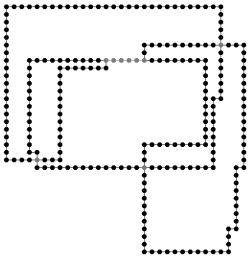
e) q, q' overlap but are disjoint at both endpoints. Then, by the same arguments as in d), both target corners exist.

CHAPTER 6

Gathering a Closed Chain of Robots on a Grid
Closed-Chain-Gather Strategy
Extended Basic&Plain Closed-Chain Robot Model







We consider the following variant of the two-dimensional gathering problem for swarms of robots: Given a swarm of n indistinguishable, point-shaped robots on a two-dimensional grid. Initially, the robots form a closed chain on the grid and must keep this connectivity during the whole process of their gathering. Connectivity means that neighboring robots of the chain need to be positioned at the same or neighboring points of the grid. In our model, gathering means to keep shortening the chain until the robots are located inside a 2×2 subgrid.

Our model is completely local (no global control, no global coordinates, no compass, no global sense of chirality, no global communication or vision, …). Each robot can see only its next constant number of left and right neighbors on the chain. This fixed constant is called the *viewing path length*. All its operations and detections are restricted to this constant number of robots. Other robots, even if located at neighboring or the same grid point, cannot be detected. Only on the basis of the relative positions of its detectable chain neighbors can a robot decide to obtain a certain state. On the basis of this state and their

local knowledge, the robots do local modifications to the chain by moving to neighboring grid points without breaking the chain. These modifications are performed without the knowledge whether they lead to a global progress or not. We call the robot model *Extended Basic&Plain Closed-Chain*. For the time model, we use the fully synchronous $\mathcal{FSYNC}$ model. For this problem, we present a gathering algorithm that needs only linear time. This result generalizes the result from [KM09], where an open chain with specified distinguishable (and fixed) endpoints is considered.

6.1 Introduction

In Chapter 5, we developed a local $\mathcal{O}(n)$ strategy for solving the general gathering of a swarm of n robots on the grid, using the *Extended Basic&Plain* robot model. In contrast to the *Basic&Plain* robot model, *Extended Basic&Plain* includes a constant memory and locally visible states. In the current chapter, we also develop and analyze a local $\mathcal{O}(n)$ gathering strategy on the grid, but using a different definition of connectivity and vision: We use the *Extended Basic&Plain Closed-Chain* robot model that we construct by modifying the *Extended Basic&Plain* robot model. We drop vision and connectivity that are solely based on L_1-distance and neighborhoods on the grid, respectively. Instead, we initially connect all robots as one arbitrary winding closed chain, while two chain neighbors have L_1-distance 1. Two robots are connected if they are neighbors on the chain and their L_1-distance is ≤ 1. We maintain this connectivity during the whole process of the gathering. Then vision is defined as follows: A robot can see other robots only along the chain and only up to a constant distance along the chain. Other robots, even if located at a short or even zero L_1-distance on the grid, are not visible to a robot. Overlappings and self-intersections of the chain are allowed, but, because of our vision definition, in general cannot be detected by the affected robots.

This significantly complicates the gathering, because robots that are located at the same position now have different viewing ranges, i.e. see different robot positions and states, and as a result clearly also behave differently.

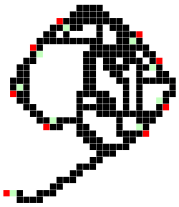
For an example of the *Extended Basic&Plain Closed-Chain* definition, look at Section 2.2.5 (page 14) where the corresponding Figure 2.3 is explained.

In our *Extended Basic&Plain Closed-Chain* robot model, gathering means to

keep shortening the chain, while maintaining the global chain connectivity until the robots have gathered. Our robot model is completely local (no global control, no global coordinates, no global compass, no global sense of chirality, no communication, no global vision, …).

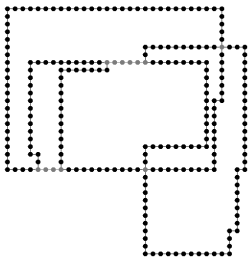
The robots only have local vision. Each of the robots can see only the subchain, consisting of itself and its next 11 chain neighbors, including their relative positions and visible states, in both directions along the chain. We call this the robot's *viewing range*, with 11 being the *viewing path length* which is the analog to the name *viewing radius* that we used for the *Basic&Plain* and *Extended Basic&Plain* robot models. The locality of our strategy is given by restricting all recognitions and actions of a robot to its constant-sized viewing range. Other robots cannot be detected by a robot, even if they are located at the same or neighboring grid points and furthermore do not restrain its movements. The robots are indistinguishable, i.e., they do not have IDs.

Concerning the robots' visualization, other than in the remaining chapters, in the current chapter's figures we draw robots located at grid points instead of at grid cells. This is only for the sake of better readability of the chain connectivity. It does not make any difference apart from that.



We use the fully synchronous time model $\mathcal{FSYNC}$. Time is subdivided into equally sized rounds of constant lengths. In every round all robots simultaneously execute their operations in the common *look-compute-move* model [CP04] which divides one operation into three steps. In the *look* step the robot gets a snapshot of the current scenario from its own perspective, restricted to its constant-sized viewing range. During the *compute* step, the robot computes its action, and eventually performs it in the *move* step.

A robot can change its position to its horizontal, vertical or diagonal neighboring grid points, while our algorithm ensures that this does not disconnect the chain. We say that the robot *hops* or performs a *hop* to the according position. A *hop* can modify the structure of the chain. Figure 6.1 shows an example for this: If two direct chain neighbors are located at the same grid point after the hop, then from then on we want them to behave like one robot. In our model, their neighborhoods are merged and one of both is removed. This *merge* operation shortens the chain by removing robots. This is also our progress measurement of the gathering. Note that robots which are located at the same grid point, but not being neighbors on the chain, are not merged/removed by



this operation.

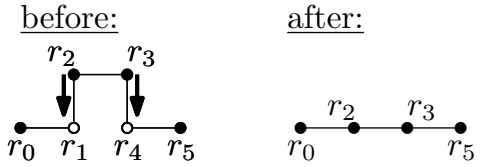


Figure 6.1: Significant example of shortening the chain (*merge*): At the same time both r_2 and r_3 hop downwards while the other robots in the figure do not move. After the hops, r_2 is located at the same position as r_1 and r_3 is located at the same position as r_4. Then the neighborhoods merge and the white robots can be removed.

We present a strategy without global information that solves the gathering problem in time $\mathcal{O}(n)$. We say that gathering is solved when all robots are located within a 2×2 square, since in our time and robot model this symmetry cannot be broken. Our result is asymptotically optimal for worst-case closed chains.

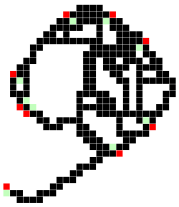
As mentioned, we want to do the gathering by shortening the chain. For the same connectivity and vision model, but for an open chain between two fixed (unmovable) endpoint robots, the first shown runtime bound for the problem of shortening the chain was $\mathcal{O}(n^2 \log(n))$ [Dyn+06]. Later, this has been improved [KM09]: In the Euclidean plane, the Hopper strategy delivers a $\sqrt{2}$-approximation of the shortest open chain in time $\mathcal{O}(n)$. Restricted to a grid, the Manhattan Hopper strategy delivers an optimal solution in time $\mathcal{O}(n)$.

Assuming that both endpoints of the open chain are located at the same position, this problem looks similar to our closed chain problem. The gathering of an open chain would furthermore be simple in general, as the endpoints are always locally distinguishable and would simply sequentially hop onto their inner neighbors. In our closed chain problem, none of the robots are distinguishable, so these solutions cannot be applied.

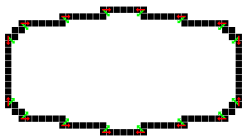
Nevertheless, as we also did in Chapter 5 we adopt the concept of movable robot states from the Hopper strategies. Roughly speaking, these strategies work as follows: A fixed endpoint of the open chain sends out a state—which we call *run* state—which then moves along the chain towards the other chain endpoint. A robot that currently has the state can execute some certain action, while the strategies ensure that for every started run the initiated action of at least one of the subsequent chain robots continues the stepwise shortening of

the chain. The authors show that this even works if multiple runs are active at the same time. So the endpoint sends out such a moving state multiple times, started every constant number of timesteps, until the robots' actions have, in the case of the Manhattan Hopper, optimally shortened the chain and reduced the number of robots to a minimal needed number. Note that though for the sake of simplicity we will say a robot r that currently owns a run state "moves its state/run to a neighboring robot r'", this is implemented the other way round: When r still owns the run, then r' can calculate that it will receive the run next, because it can see the active run state (including its moving direction) of r. So, r' sets its run state on its own without requiring further information from r.

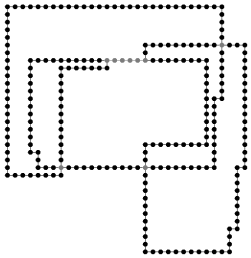
Since the robots themselves are indistinguishable in our closed chain model, no robot has the given special role to generate new run states. We instead do a distinction depending on the relative positions of the constant number of chain neighbors within the robots' local viewing ranges. Using this criterion, the robots elect themselves to obtain the special role of sending out the states. Using this approach, we have to deal with several difficulties. Cf. Figure 6.2.



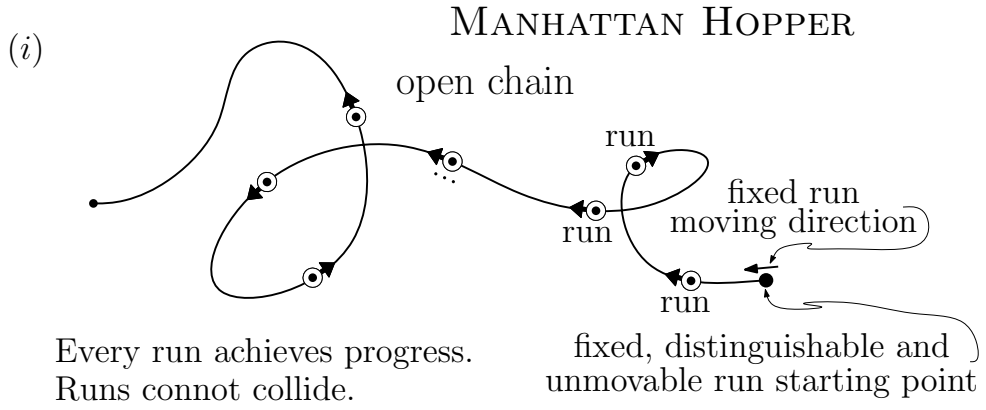
- At the same time, more than just one robot obtain the special role.
- New runs obtain different moving directions along the chain. Collisions (e.g., S_1, S_2) must be solved.
- Only some of the runs will actually achieve a shortening of the chain, but the robots cannot detect this, because of the restricted local viewing ranges.



- The ones that do not achieve a shortening of the chain shall not hinder the work of the others.
- Because robots are moving, at different times different robots obtain the special role. So new states start from different robots.



As already mentioned, in the chain-based vision and connectivity model that is used in the current chapter, robots that are located close to each other or even at the same position in general see completely different, independent viewing ranges, and so they behave completely differently and independent of each other. This clearly is a very destructive behavior for the objective to let all



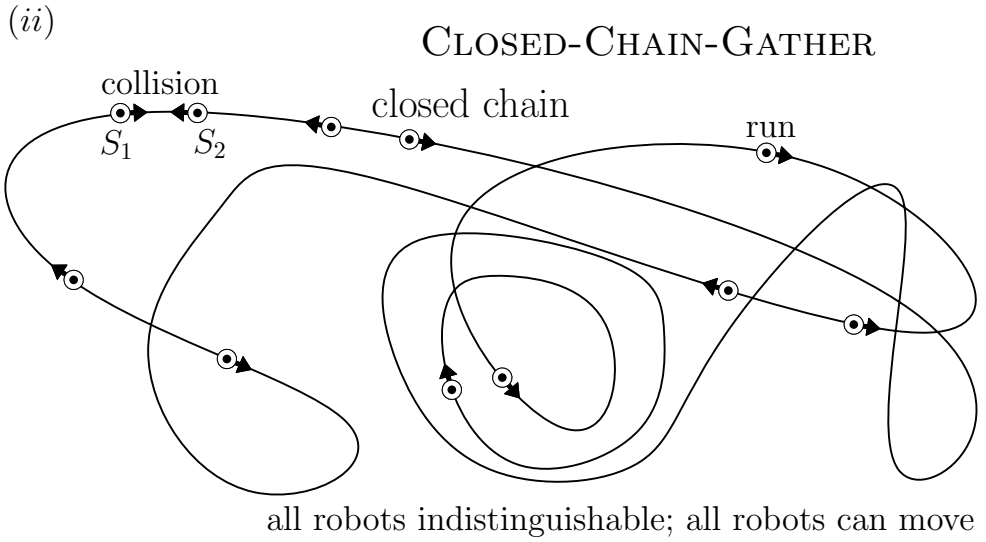


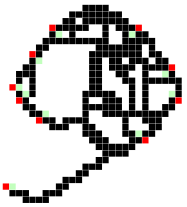
Figure 6.2: Symbolic comparison of the (*i*): Manhattan Hopper and the (*ii*): Closed-Chain-Gather strategy.; The approach of movable run states cannot be adapted easily for closed chains.

robots move to the same position (gathering). (For example, in our Oblivious-Gather and Grid-Gather strategies, arbitrary robots always merge to one robot if they are located at the same position that then always means an irreversible gathering progress.) As consequence, the correctness and running time proofs of the current strategy become much more challenging. Furthermore, these proofs require significantly different approaches than the corresponding proofs of the Grid-Gather strategy, which also uses the concept of movable robot states.

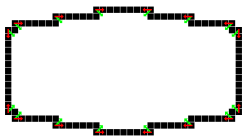
6.2 Basic Idea of the Algorithm

Our measurement of progress is the shortening of the chain. Then, after $n - 1$ shortenings the gathering is done. In our model, the chain can be shortened if two chain neighbors are located at the same grid point. Then, one of them is removed and for the other one the neighborhoods of both are combined and assigned to it, which results in the shortening of the chain.

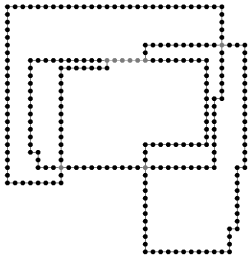
Because a robot's viewing range is restricted to just a constant size, these shortenings cannot be performed easily in general. Our algorithm performs two basic operations. More precisely, we have to deal with two cases:



1. Some neighboring robots perform a single hop such that afterwards at least two chain neighbors are located at the same position, while preserving the chain connectivity. Afterwards, the shortening as explained above can be performed. This is a so-called *merge* and further discussed in Section 6.2.1.



2. If on some subchains merges are impossible, we perform so-called reshapements to reshape the chain in order to prepare merges in succeeding steps. These are explained in Section 6.3.1.



In Section 6.2.1, we now start with case 1. For the sake of simpler descriptions, we use terms like *horizontal, vertical, downwards, left, …* Since our robots do not have a common sense for this, the descriptions/figures are also to be understood in a mirrored or rotated manner.

6.2.1 Merges

As introduced, the chain can be shortened if two chain neighbors are located at the same grid point. Figure 6.3 shows how our algorithm manages that two chain neighbors get located at the same grid point. In the figure, the black robots hop downwards simultaneously. Afterwards, assuming that the length k of the black subchain is bigger than 1, the both outermost of them are located at the same grid point as the bordering white robots. Because at both ends the robots are chain neighbors, we can remove the white ones without breaking the chain. If $k = 1$, then after the hop also both white robots can be removed without breaking the connectivity of the chain. We call these operations *merge operations* or shorter: *merges*. Merge operations can be performed only if all participating robots can see all black and white robots. This is necessary, because otherwise not all of the black robots know that they must perform a hop in order to prevent the chain from breaking. So the length k especially cannot be larger than a robot's constant viewing path length.

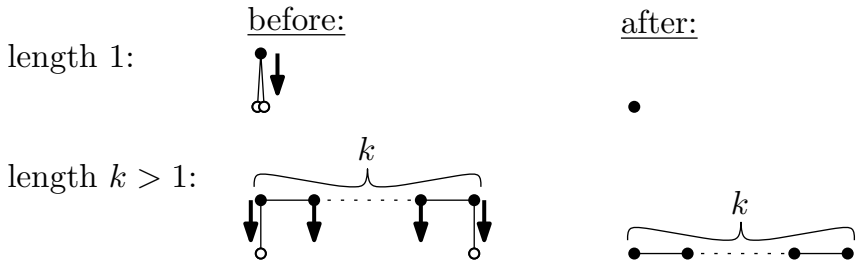


Figure 6.3: Subchains that allow progress hops (*merge operations*). $k = 1$: Here, the white robots are actually located at the same grid point and are drawn with a small distance just for presentation reasons. The value of k indicates the length of the black subchain. k is upper bounded by a robot's constant viewing path length.

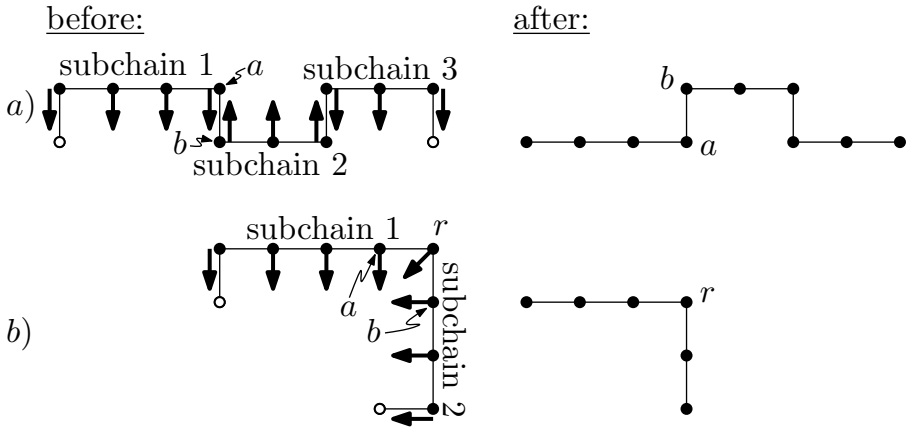
It may happen that at the same time more than just one merge can be performed on different parts of the chain. Then, two cases need a closer look (We refer to the black and white robots of Figure 6.3.):

1. The merge subchains, including the white robots, overlap by two robots.
2. The merge subchains, including the white robots, overlap by three robots.

An example for 2) is shown in Figure 6.4.*b*). Here, the robot r belongs to the black robots of subchain 1 and 2. This requires an additional rule, because

concerning subchain 1 r would hop downwards, but concerning subchain 2 it would hop to the left. Instead, we let r perform a diagonal hop to the lower left, while the other black robots perform their usual hops. Afterwards, r, a, b are located at the same position and a, b are removed without breaking the chain.

An example for 1) can be seen in Figure 6.4.*a*). Here, all black robots perform the same hop as the black ones in Figure 6.3. The difference is that afterwards, for example, robot a is not located at the same position as b and vice versa. So, there the chain cannot be shortened. If as in the example of Figure 6.4.*a*) the outermost white robots do not move, then here the shortening can be performed. For contradiction, we assume that the whole chain consists only



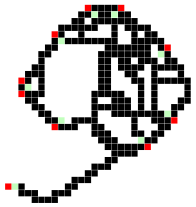
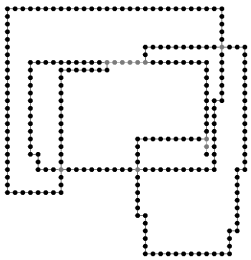


Figure 6.4: Cases for merges of subchains which are not node disjoint. (significant examples)



of merge subchains. Figure 6.5: *a*) If only merges of length > 1 exist then, starting from s, the chain could not be closed using only overlappings of Type 1). Overlappings of Type 2) (marked by the dashed curve) must also exist. *b*) In contrast, if merges of length 1 are allowed, overlappings of Type 2) are not needed. In the algorithm, in two steps, the robots separately first perform only merges of length 1 and afterwards only merges of length > 1, so that always robots can be removed. In summary, because all cases of merge operations lead to the removal of robots, i.e., shortening of the chain, all satisfy our measurement of progress.

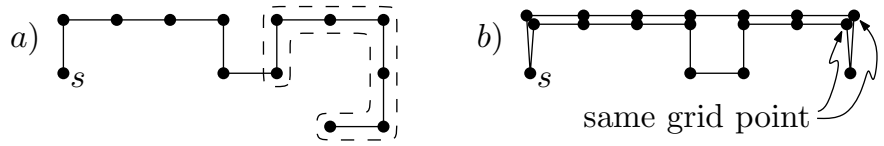


Figure 6.5: Chains, consisting only of merge subchains.

6.2.2 Reshapement of the chain, done by runners

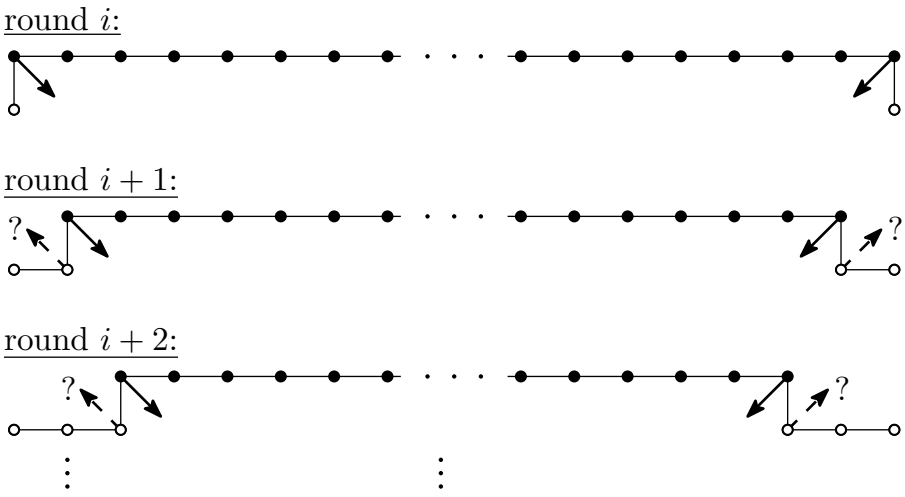


Figure 6.6: If the length of the black subchain (the value of k in Figure 6.3) is larger than the robots' viewing path length, we shorten it by letting the outermost black robots perform diagonal hops. We indicate such hops by diagonal arrows.

If because of the robot's restricted viewing ranges a *merge* cannot be performed anywhere on the whole chain, then we call the chain a *Mergeless Chain*. We perform so-called *reshapements* to reshape the chain in order to prepare merges in succeeding steps: We continuously let the outermost robots of the black subchain perform diagonal hops in order to make it shorter (cf. Figure 6.6) until it becomes short enough for enabling the operation of Figure 6.3. We say that these hops *reshape* the subchain. This task introduces two major challenges:

1. When the outermost black robots of Figure 6.6 decide to start the reshapement (round i), within their restricted viewing range they do not know if their task of performing diagonal hops will lead to merges.

2. In the following rounds, the subchain which an outermost black robot can see in general locally looks the same as the one that is visible to its white neighbor. So, we have to ensure that the reshapement is continued by the outermost black robots instead of by their white neighbors.

In the following, we tackle these challenges. For this, we first introduce a certain state of a robot, which we call the *run* state. We call a robot with an active run state a *runner*. Robots can achieve this state in two different ways:

1. (*start run state*) If the local subchain within a robot's viewing range has a certain shape, then the robot decides on its own to generate the run state. We say that such a robot starts a run. On the basis of the shape of the local subchain, the run state gets a fixed moving direction along the chain. A robot can start and store up to two run states at the same time. Figure 6.7 shows how the local shapes must look like.

2. (*move run state*) A runner $R(S)$ can move the run state S to its chain neighbor r' in the moving direction of S. We say that the run state has moved from $R(S)$ to r', while its moving direction initially set in 1) always remains unchanged. Afterwards, r' is identified by $R(S)$.

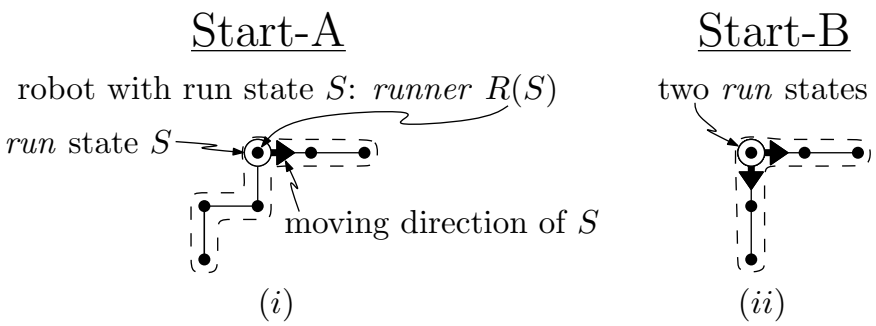
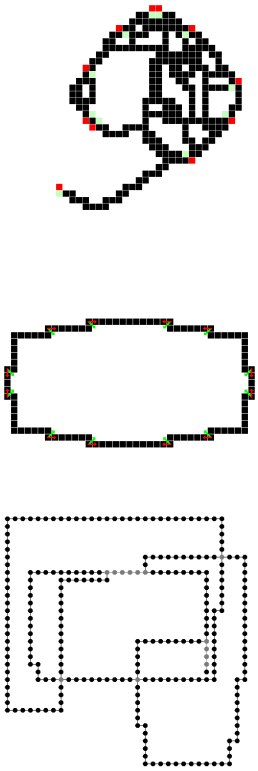


Figure 6.7: The encircled robots decide to start the run states, only on the basis of the shape of the marked subchain within their viewing range. The large arrow heads indicate the moving direction of the runs. This is the notation we will use for marking a runner. $R(S)$ identifies the runner/robot which currently has the run state S. In (ii), the encircled robot is the endpoint of a horizontally and a vertically aligned subchain at the same time. Here, we must start two runs moving in both directions along the chain.



Once a run state has been started in 1), 2) is executed in each of the following rounds. This means that the run moves along the chain at constant speed and

Figure 6.8: If in round i the local subchain looks like this, then the runner performs a diagonal hop and the run state moves to the next robot in the moving direction.

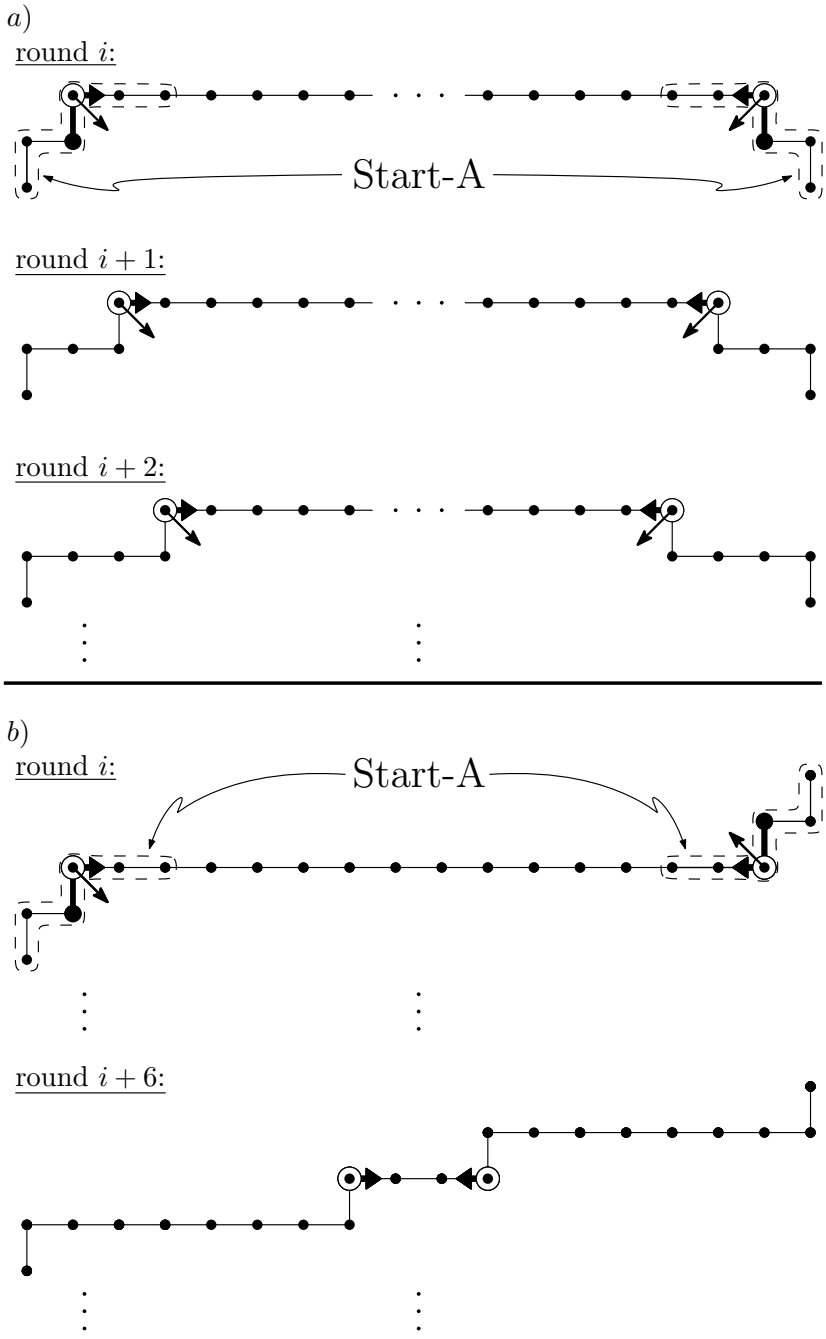
in the initially settled moving direction. A runner can perform a diagonal reshapement hop (and afterwards move the run state to its chain neighbor in moving direction) if the chain locally looks like in Figure 6.8 (round i). Figure 6.9.a) shows sequent operations. This solves the problem we have had in Figure 6.6, where it was impossible to locally decide by the robots which of them has to perform the reshapement hop. But not all run pairs started at the endpoints of a horizontally aligned subchain actually do enable a merge. We distinguish *good pairs* and *non-good pairs*.

good pairs: (Cf. Figure 6.9.a)) If the exterior neighbors (the fat robots in the figure) of the newly started run pair are both located on the same side of the subchain, then the run pair is called a *good pair*. These pairs enable a merge if they have been moving close enough together and then terminate.

non-good pairs: Figure 6.9.b) shows the opposite case, i.e., the fat robots are located at different sides of the subchain. These pairs do not enable a merge. We let the runs of such pairs pass along each other. Figure 6.10 shows how this is performed: At the time when their distance (i.e., the number of edges on the subchain connecting both) is 3 or less, they only keep moving along the chain, but the runners do not perform reshapement hops. This is repeated until S_1 is located at its target robot c_2 (then also S_2 is located at its target c_1). We call this the *run passing* operation. Afterwards, the normal reshapement operations are continued. Note that depending on whether the distance between S_1 and S_2 was odd or even, it happens that during this passing process at one time both runs are located at the same robot. Then, this robot handles both runs separately according to their movement directions.

6.2.3 Parallelizing runs: *Pipelining*

The total process of gathering needs the work of many good pairs. For sake of a short running time, we let some of them work in parallel. Every constant



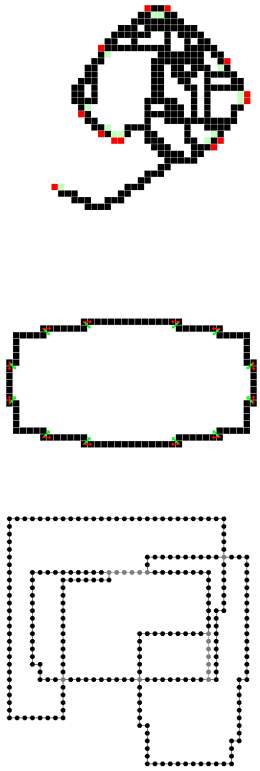


Figure 6.9: Run pairs, started at both ends of horizontally aligned subchains. a, b): In round i, the new runs start (cf. Figure 6.7). Afterwards, the actions of Figure 6.8 are repeatedly executed. The runs move closer and closer together. a): The run pair is a *good pair*. If the runs have moved close enough, then a merge can be performed. b): No merge can be performed. Then the runs just pass along each other. (Details: Figure 6.10)

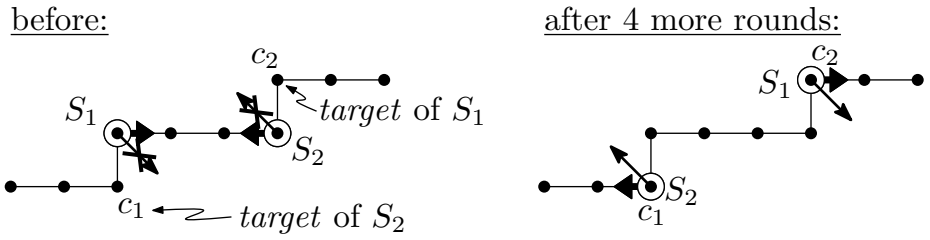


Figure 6.10: *Run passing* operation: The runs S_1, S_2 do not enable a merge. If their distance along the chain is less or equal 3, then they pass each other by only keeping moving but without making the runners perform diagonal hops. Afterwards, i.e., when S_1, S_2 have reached their target robots/corners c_2, c_1, they return to normal operation.

number of $L = 15$ rounds, all robots simultaneously check if they can start new runs (cf. Figure 6.7) and if so, they do so. We will show that this procedure ensures that even if multiple good pairs are nested into each other, different good pairs will enable different merges. This is what we call *pipelining*. Figure 6.11

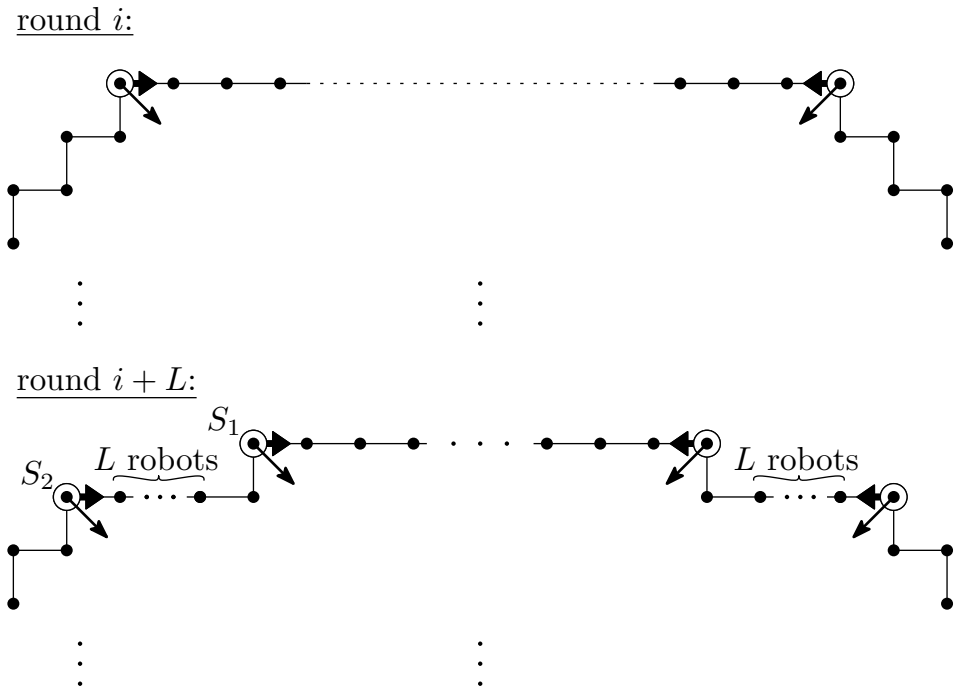


Figure 6.11: Pipelining of runs. New runs are started every $L = 15$ rounds.

shows an example of this. Because S_1, S_2 are moving in the same direction, we

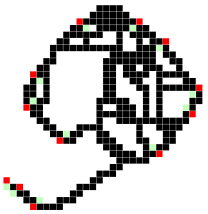
call them *sequent* runs, while relative to their moving direction S_1 is located *in front of* S_2. The *distance* between them equals the number of edges on the subchain connecting both.

As runs are moving with constant speed, the runs of the inner good pair will meet first and, as the result, first enable a merge and stop. Then, obviously the outer good pair will also enable a (different) merge, some rounds later.

6.2.4 Stopping runs

In order to ensure that the pipelining works correctly, we let a runner $R(S)$ stop/terminate its run S if one of the following conditions is true:

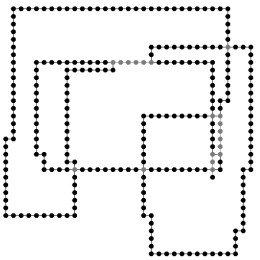
1. It can see the next sequent run in front of it. (This happens if sequent runs have come too close to each other, e.g., because of merge operations.)
2. It can see the end of the horizontally aligned subchain it is located on, in front of it.
3. It was part of a merge operation.
4. While it performs the run passing operation of Figure 6.10, the target corner is removed. (This can happen because of a merge operation.)



We give some more detailed explanations concerning some of the above conditions:

1) For example, because of merges two sequent runs may come too close to each other, which then might hinder the pipelining. The affected runners can detect this on their own. The criterion for this is that the next sequent run in front of them becomes visible. Then, the termination condition 1) matches and the run behind stops.

4) We assume that in Figure 6.10 to the right of robot c_2 another run S_3, moving in the same direction as S_2, is located. During the rounds in which S_1 and S_2 are performing their run passing operation, S_3 keeps moving towards c_2. Now it may happen that because of the reshapements of S_3, c_2 becomes part of a merge operation. Then, c_2 would hop downwards such that the corner shape does not exist anymore. Because this corner has been the target of S_1, S_1 could not continue its reshapements after the run passing, so it terminates.



6.2.5 Correctness and running time

If a merge can be performed every round, then the time needed for the gathering is obviously upper bounded by n, with n being the number of robots. If no merge can be performed, then the shape of the chain is reshaped by runs. We have shown that good pairs enable merges. So we need to show that if no merge is possible anywhere on the whole chain, always at least one new good pair can be started.

A single good pair needs at most n rounds until a merge can be performed. In Section 6.2.3, we have already noticed that if new good pairs are started every $L = 15$ rounds, then different good pairs lead to different merges. The last good pair is started at round $L \cdot n$ and finishes its work after at most n further rounds. As L is a constant, we then get to the total linear running time $\in \mathcal{O}(n)$.

6.3 Algorithm in Detail

Now, we explain the complete strategy. In Section 6.2, we let runs move only along strictly horizontally aligned subchains. But such subchains do not always exist. We need to generalize them to so-called *quasi lines* (see Definition 6.1). The main difference to the description of Section 6.2 is that we now let runs move along such *quasi lines*. Figure 6.12 gives an example of a quasi line.

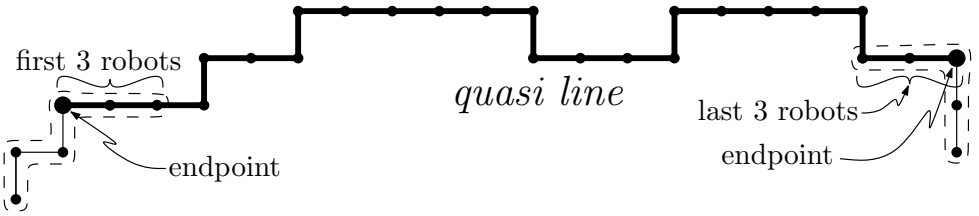


Figure 6.12: Example of a quasi line. The fat robots are its endpoints.

Definition 6.1 (quasi line)**.** We call a subchain a horizontal *quasi line* if the following points hold:

1. At least its first and last three robots are horizontally aligned.
2. All its subchains of horizontally aligned robots contain at least three robots.

3. All its subchains of vertically aligned robots contain at most two robots.

In a Mergeless Chain, at both ends a subchain Start-A or Start-B (cf. Figure 6.7) in a matching rotation or reflection occurs. (If the chain is not mergeless, then the subchains outside the quasi line's endpoints may also have other shapes than these.)

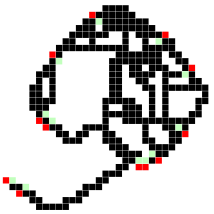
The definition of a vertical *quasi line* follows analogously.

In Figure 6.12, the fat subchain connecting and including the fat robots at its endpoints is called a quasi line.

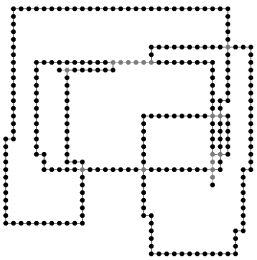
Having introduced quasi lines we now have to analyze, how this affects the *merges*, *runs* and *pipelining* we have explained in Section 6.2.

Merges remain exactly the same as in the basic description (Section 6.2.1), i.e., they are only performed with black subchains, solely consisting of strictly horizontally aligned robots as shown in Figure 6.3. So, we can continue with *Mergeless Chains* and the analog to Section 6.2.2.

6.3.1 Reshapement of the chain, done by runners



The main approach remains the same as in Section 6.2.2: i.e., if in Figure 6.3 the black subchain is longer than the robots' viewing path length, we use reshapement hops of runners for shortening this black subchain until a merge becomes possible. We start new runs at the same subchains as in the basic description (cf. Start-A and Start-B in Figure 6.7). The only difference is that now these subchains are connected by a quasi line. Because of this, as one can see in the example of Figure 6.15, the runs now have to move several steps along the chain until arriving at the endpoints of the subchain, bordered in the figure, which needs to be shortened for performing the merge. For this movement, depending on the local shape of the subchain, we require additional run operations: Figure 6.13.*a*) (OP-A) is the basic operation, while *b*) (OP-B) and *c*) (OP-C) are the new ones.



a) OP-A: The runner and at least the next 3 robots are located on a straight line. Here, the runner first performs a diagonal hop, then moves the run to the next robot.

b) OP-B: The runner and only the next 2 robots are located on a straight line.

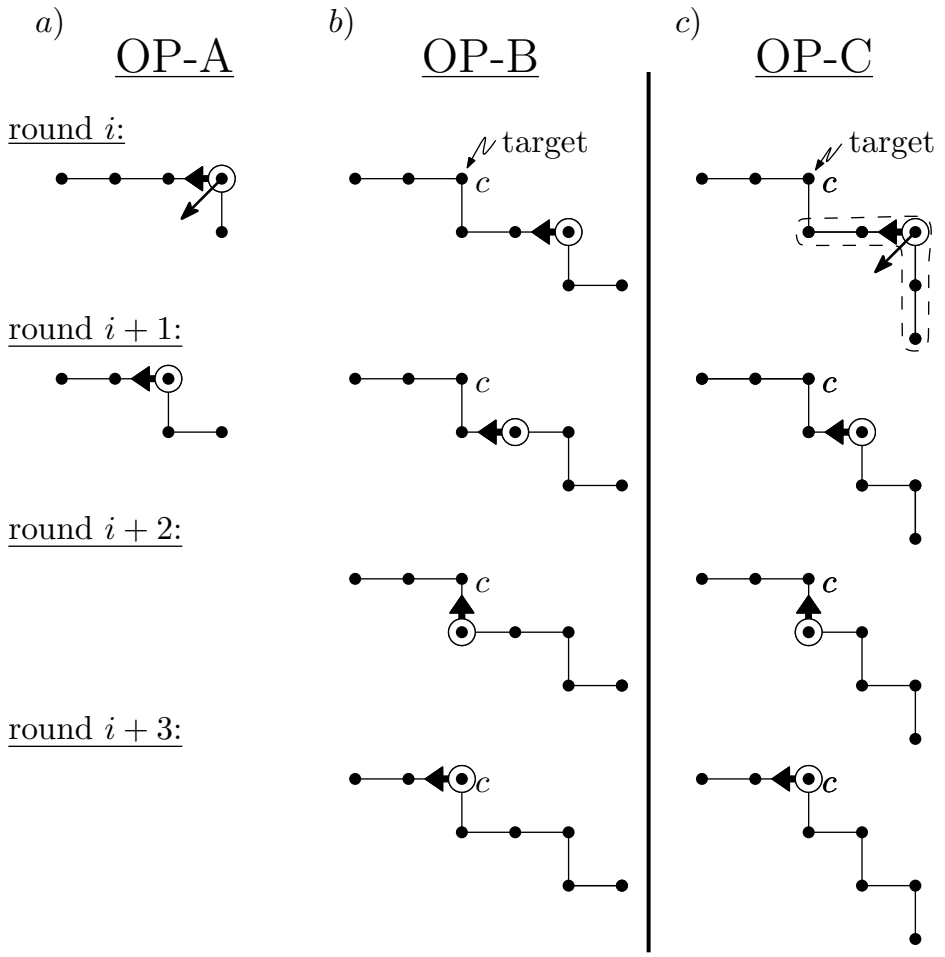


Figure 6.13: *a*) (OP-A): Reshapement by a runner. The operation takes only one round. *b*) (OP-B): No diagonal hops are performed until the target corner *c* is reached. *c*) (OP-C): Special case when new runs start: First, perform one diagonal hop, then no diagonal hops until the target corner *c* is reached.

Then, for 3 times the runners just move the run to the next robot without any diagonal hops. Afterwards, it is located at the target corner c.

c) OP-C: This one is needed at most once for a new run if started at the subchain of Figure 6.7.(*ii*) (page 101).

New runs always start pairwise at both endpoints of a quasi line. *Good pairs* of runs are defined analogously to Section 6.2.2 by the relative position, concerning the quasi line, of the outer chain neighbors (the fat robots) of the endpoints of the quasi line (cf. Figure 6.14). We will show that also with quasi

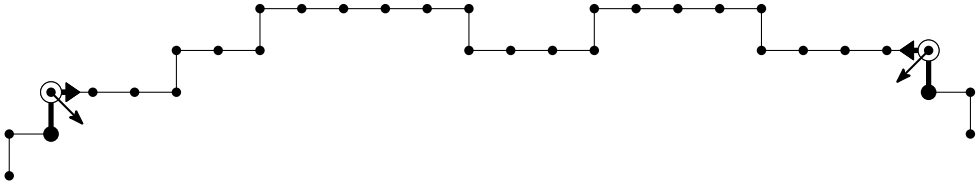
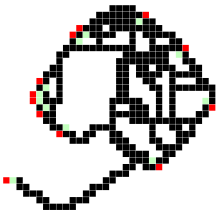
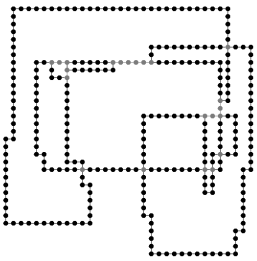


Figure 6.14: Good pairs of runs, connected by a quasi line. The runs are a good pair if the outer chain neighbors (the fat robots) are both located either downwards or both upwards, i.e., on the same side of the quasi line.



lines, good pairs enable merges. And as before in Section 6.2.2, not all run pairs are good pairs. Figure 6.15 shows an example of a good pair on a quasi line: The bordered subchain will be reshaped for performing a merge. Until round $i + 4$, the runners execute only the basic operation OP-A (Figure 6.13.*a*)). Afterwards, the runner $R(S_r)$ starts operation OP-B, while $R(S_\ell)$ still executes OP-A. After operation OP-B has been processed completely, S_r finally is located at the right end of the bordered subchain (round $i + 8$). Now its runners start shortening this subchain by executing OP-A until the merge can be performed. Afterwards, S_ℓ is still active and keeps moving and will stop at the latest when an endpoint of the quasi line becomes visible.

If two runs that do not enable a merge meet each other, we execute the run passing operation in a more generalized variant. As some of the operations of Figure 6.13 now take more than just a single round, it may happen that such an operation becomes interrupted by the run passing. Then, the target corners for the run passing are settled with respect to the situation when the interrupted operation has been started. Figure 6.16 shows an example: In round i the runner $R(S_1)$ starts the execution of operation OP-B of Figure 6.13. In round



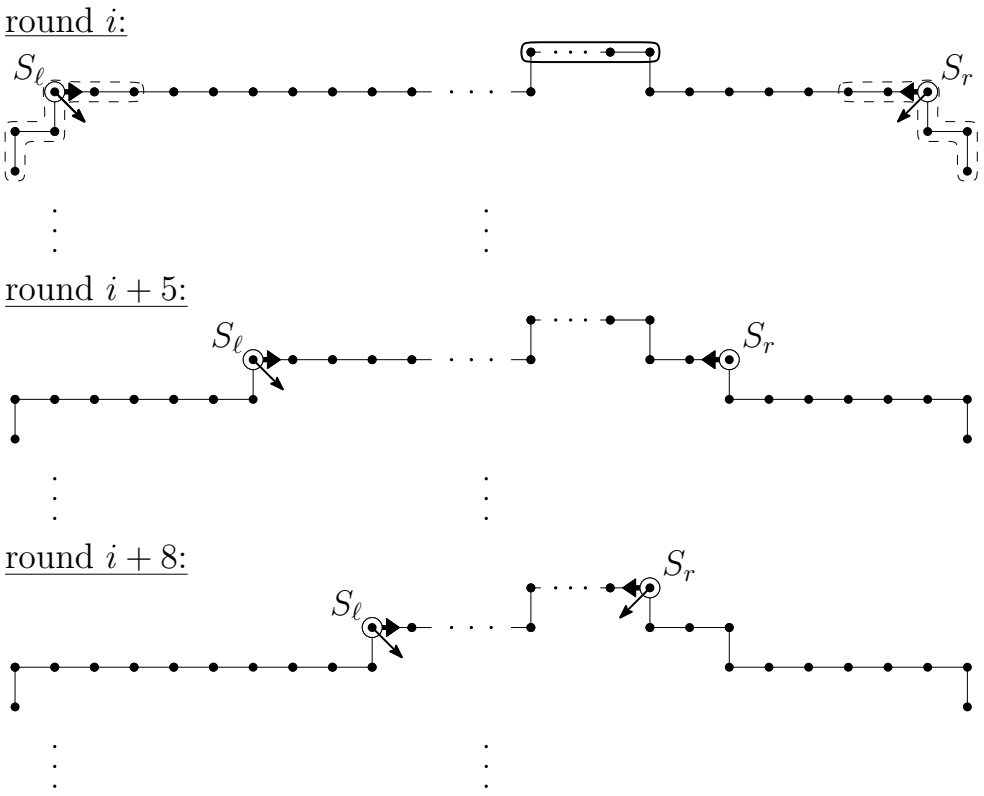


Figure 6.15: A good pair, started on a quasi line in order to shorten the bordered subchain. Several and different run operations are needed.

$i+2$, this operation is still not finished, but the distance between S_1 and S_2 is 3 such that their runners both have to start the run passing operation. Then, the target corner of S_2 is the corner c_1. The target of S_1 as before is c_2. Similar to the basic run passing, both runs do not perform any reshapement hops before arriving at their target corners. Other cases, e.g., if the runners of both runs execute operation b), are analog to this one.

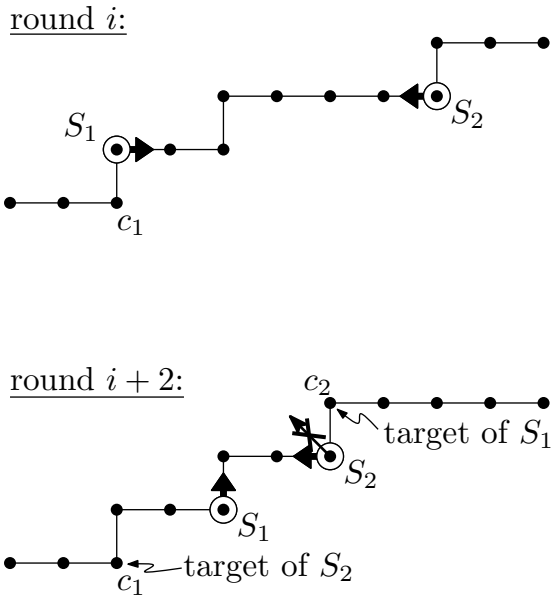


Figure 6.16: Runs passing along each other while the runner $R(S_1)$ is executing the operation OP-B (Figure 6.13.b)).

6.3.2 Parallelizing runs: *Pipelining*

The pipelining works the same as in the basic explanation in Section 6.2.3.

6.3.3 Stopping runs

The basic stop/termination conditions for runs (Section 6.2.4) have to be extended for working on quasi lines. We let a runner stop/terminate its run if one of the conditions of Table 6.1 is true. In summary, all robots synchronously execute the algorithm, shown in Figure 6.17.

Every robot r every round checks the following three steps:

1. Merge: (In each round executed two times: First execution: Robots perform merges only if length = 1. Second execution: The robots perform merges only if length > 1.)
   If r detects a possible merge within its viewing range then
   - If r is one of the black robots in Figure 6.3, it hops downwards.
   - If afterwards r is located at the same position as one of the white robots, the white one is removed without breaking the chain. (If it is located at the same position as both white robots, then both white ones are removed.)
2. Run Operations: If r is a *runner*, then
   a) Its run terminates/stops if any of the conditions of Table 6.1 is true.
   b) Runner's Movement and Reshapement
      - Run passing:
        - If r is currently in progress of executing the run passing operation (Figure 6.10 respectively 6.16), then this operation is continued.
        - Else, if r can see a run in front of it, such that both are moving towards each other and the distance between them is less or equal 3, then r starts the run passing operation.
      - If r is not in progress of passing, then
        - If r is in progress of executing a run operation OP-B or OP-C (Figure 6.13.b, c)), which takes more than one round, then this one is continued.
        - Else: r executes the matching new run operation of Figure 6.13.
3. Start new runs: Every $(L = 15)$th round, r checks if it can start a new run:
   If r is one of the encircled robots of Figure 6.7.(i) (Start-A) respectively (ii) (Start-B), then it starts one respectively two runs.

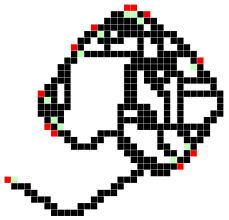
Figure 6.17: The algorithm.

A runner stops/terminates its run if at least one of the following conditions is true:

1. It was part of a merge operation.
2. It can see the next sequent run in front of it. (This happens if sequent runs have come too close to each other, e.g., because of merge operations.)
3. It can see the endpoint of the quasi line in front of it.
4. While it performs the run passing operation of Figure 6.10, the target corner is removed. (This can happen because of a merge operation.)
5. While it performs the operation OP-B or OP-C of Figure 6.13.b, c), the target corner is removed. (This can happen because of a merge operation.)

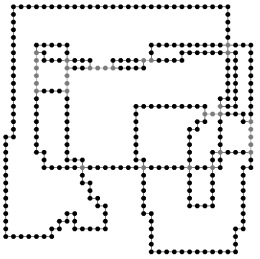
Table 6.1: Conditions which let a run terminate.

6.4 Correctness and Running Time



In this section, our goal is the proof of the correctness and the linear running time (Theorem 6.4). In our main approach, we want to enable merges by good pairs if else no merge could be performed. The following two lemmas prove that this actually works. They are the base for the proof of the theorem. The proofs of the lemmas can be found in Section 6.4.1 and 6.4.2.

Because of the local vision, new runs and maybe also good pairs are started every $L = 15$ rounds, regardless of whether or not at some other location on the chain merges can be performed. In the following analysis, we will argue only with new good pairs which are started if during the last $L - 1$ and the current round on the whole chain no merge has been performed. We distinguish such good pairs from others by calling them *progress pairs*.



Lemma 6.2. *Every $L = 15$ rounds either a merge has been performed or else a new progress pair is started.*

Lemma 6.3. *For progress pairs the following properties hold.*

a) Every progress pair enables a merge (after at most n rounds).

b) Different progress pairs enable different merges.

Using these two lemmas, we can now prove the total linear running time.

Theorem 6.4. *Given a closed chain of n robots, then after $\mathcal{O}(n)$ rounds gathering is done. This is asymptotically optimal.*

Proof. We subdivide time into intervals of lengths L, where L denotes a number of rounds. Merges can be performed during at most n such intervals, because every merge removes at least one robot. In all other intervals a new progress pair starts (Lemma 6.2). Each of these progress pairs leads to merge (Lemma 6.3.*a*)). Because no two of them lead to the same merge (Lemma 6.3.*b*)), the number of intervals without merges is also upper bounded by n.

By Lemma 6.3, a progress pair needs at most n rounds until it has led to a merge. We assume the worst case that in the last of the $2 \cdot n$ intervals the last progress pair was started. Then the total running time is upper bounded by $2n \cdot L \ + n$, which proves the upper bound of the theorem, because L is a constant.

In our model, the diameter of the initial configuration provides the worst-case lower bound $\Omega(n)$ for any gathering strategy. □

6.4.1 Proof of Lemma 6.2

Proof. The lemma assumes that the chain is a *Mergeless Chain*. Although our viewing path length also allows larger values, for the proof we assume that merges are only possible up to the length 2 (cf. Figure 6.3 on page 98). This suffices, because if a chain is a Mergeless Chain for a bigger length, it also is a Mergeless Chain for shorter lengths.

As by definition all horizontal subchains of a horizontal quasi line consist of at least 3 robots, no merge can be performed on them. In order to connect two horizontal or two vertical quasi lines without enabling a merge, they must be connected by so-called *stairways*. Stairways can have arbitrary length and are subchains of alternating left and right turns. Figure 6.18 shows an example. All differently shaped connecting subchains would allow merges. If a horizontal

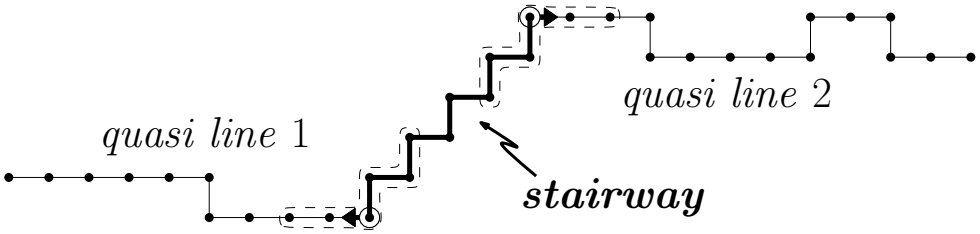
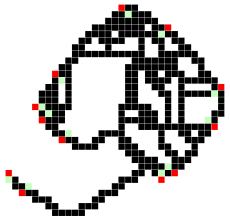


Figure 6.18: Two quasi lines, connected by a *stairway*.

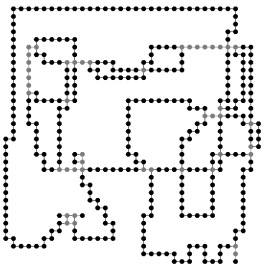
and a vertical quasi line are connected, then this can also be done without any stairway.

The above construction exactly leads to the run starting subchains (i, ii) (Start-A, Start-B) of Figure 6.7 (page 101). In Figure 6.18, such subchains are bordered by dashed curves. So new runs are always started at the endpoints of quasi lines.

For being able to close the chain, there must exist both horizontal and vertical quasi lines. We start on the left of Figure 6.19 at a robot s where a vertical and a horizontal quasi line are neighbors. For simplicity, we assume that the stairways connecting the quasi lines in the figure are of minimum length, and we symbolize quasi lines by dashed line segments. The fat robots correspond to the fat robots of Figure 6.14, i.e., they are the outer chain neighbors of quasi lines. As in Figure 6.14, the run pairs are good pairs if the fat robots are both located on the same side. So, if no good pair existed, they must instead lie on alternating sides of the quasi lines. Figure 6.19 shows how a sequence of horizontal quasi lines then must look like. Because we want to close the chain, we need a second vertical quasi line. If no merge is possible, this quasi line must point upwards, i.e., in an opposite direction than the first one. In this proof, we show by contradiction that if this always is the case, then the chain cannot be closed.



In Figure 6.20, we have grouped subchains of connected horizontal respectively vertical quasi lines and symbolize such a group by a dashed curve. For our argumentation, we take the robot s, which is the first robot of the first horizontal quasi line, as the starting point. We define an *orientation vector* which points from the second last robot of the last vertical quasi line to the second robot of the subsequent horizontal one. (If between these two quasi



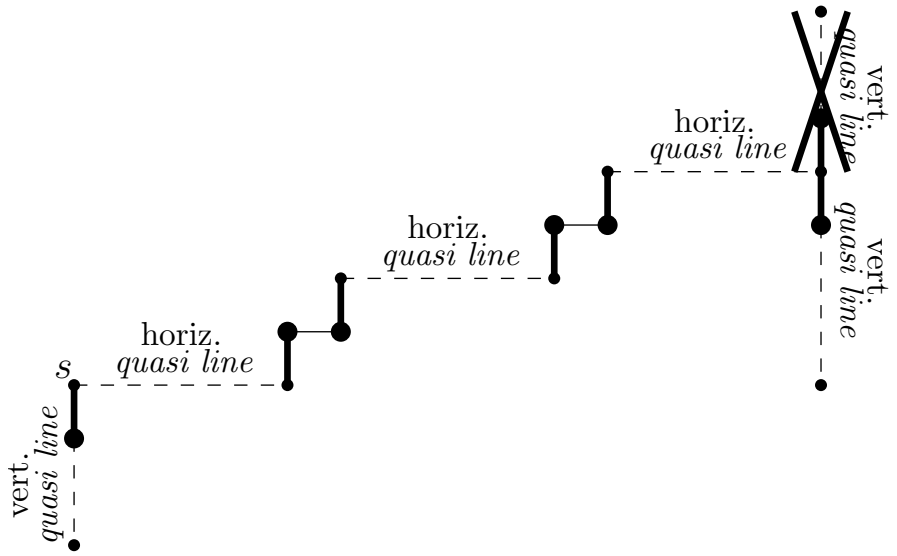


Figure 6.19: Proof idea: It is impossible that always both vertical quasi lines point to opposite directions. Then, in the example at the last horizontal quasi line, a good pair starts.

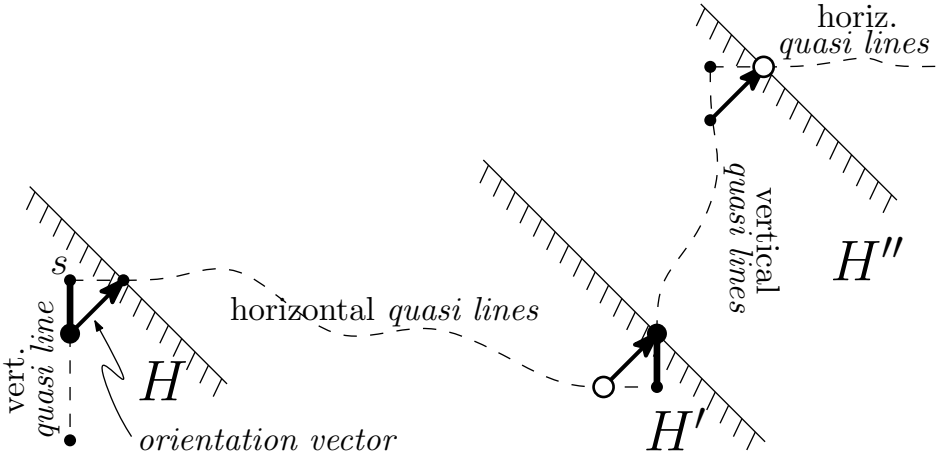
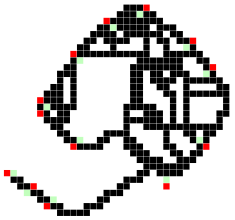


Figure 6.20: In a Mergeless Chain, good pairs do always exist.

lines a stairway exists, then the orientation vector becomes longer than in the figure, but still points to the same direction.)

The orientation vector defines the half-plane H by orthogonally pointing from the interior to its boundary. In order to close the chain, we must return to s and for this, reenter H. This cannot be done using only horizontal quasi lines. So, we look at the point where the first vertical quasi line of the next group of vertical quasi lines occurs. Here, the orientation vector and the half-plane H' are defined analogously to the previous ones. Because we assumed that the fat black robots lie on different sides of the horizontal quasi lines, the vertical quasi lines must point upwards. Then, the orientation vectors of H and H' are parallel and $H \subset H'$. If we continue the argumentation for the vertical quasi lines, we have to ensure that now the fat white robots are located at different sides. Then the following horizontal quasi lines must point to the right again. Then again the orientation vectors are parallel and we get $H \subset H' \subset H''$. Continuing this construction, we never get back to the robot s and so can never close the chain. So, a good pair must exist. Because we have assumed that the chain has been a Mergeless Chain also during the previous $L - 1$ rounds, this good pair is a progress pair. □

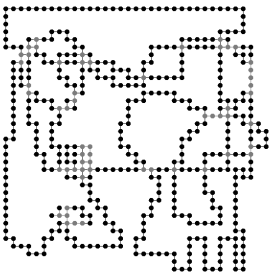


6.4.2 Proof of Lemma 6.3

For the proof, we need the run invariants of Lemma 6.5.

Lemma 6.5. *The value $L = 15$ and the value 11 for the viewing path length ensure that for every run S until it terminates, the following invariants hold:*

1. *Every round, S moves one robot further in moving direction.*
2. *After the first three rounds after its start S is always located on a quasi line (i.e., the reshapements of its runner do not violate the quasi line definition 6.1.).*
3. *S cannot see other sequent runs in front of it.*
4. *S is either in progress of passing along another run or the runner $R(S)$ executes one of the operations OP-A, OP-B, OP-C of Figure 6.13.*
5. *Good pairs stay being good pairs.*



Now we can prove Lemma 6.3. The proof of the required Lemma 6.5 is provided subsequently.

Proof. a) : At the time a progress pair S, S' is started, the subchain connecting both is a quasi line. Because of Lemma 6.5.2), this also does not change if other runs are located on this subchain. And also merges preserve the quasi line properties. So, if not stopped, S and S' keep moving towards each other (Lemma 6.5.1)). Because of Lemma 6.5.5) a merge can be performed at the latest when they meet, which takes at most n rounds.

It remains to show that S, S' are not stopped by the algorithm's termination conditions (Table 6.1) before the merge could be performed. In the following, we check all these conditions: 2): By definition of a progress pair, no merge has been performed since the last time new runs have been started. This means that the distance between S and its next sequent runs in front of it is at least $L - 3$ (cf. the proof of Lemma 6.5.4)) which is bigger than the viewing path length. (The same holds for S'.) So the runs of a progress pair cannot be stopped by this termination condition.

3): If a run of a progress pair can see an endpoint of the quasi line in front of it, then the other run of the progress pair must have previously been stopped. Because a run of a progress pair cannot be stopped by condition 2), it instead must have been stopped because of a merge. So, a termination because of condition 3) is allowed.

4, 5): If a run S of a progress pair is stopped because of this, then its target corner was removed. This could either have happened because of the reshapement of a sequent run or because of a merge operation. Because we have ensured the minimum distance of two sequent runs to be large enough, the first case cannot have happened. So, a merge must have been the reason. Then, this merge must have been enabled by another run $S^\star$, moving towards S. If $S^\star$ is the partner run of S, then stopping is allowed. Else, because progress pairs are nested into each other, $S^\star$ either was not a run of a progress pair or its partner run has previously been stopped. In the latter case, because runs of progress pairs cannot be stopped because of condition 2), the run must have been stopped by an earlier merge. In both cases, the current merge can be credited to the progress pair of S.

1): By the same arguments, this merge does not need to also be credited to a

different progress pair. This then also proves *b*) of the lemma. □

Now we provide the omitted proof of Lemma 6.5.

Proof of Lemma 6.5.

Proof. 1): This directly follows by the run definition in Section 6.3.1.

2): Cf. Figure 6.13 on page 108. *a*) (OP-A) ensures that on horizontal quasi lines only horizontal subchains of lengths > 2 are shortened and vertical subchains remain unextended. *b*) (OP-B) does not reshape. *c*) (OP-C) can be executed only during the first three rounds after S has been started.

3): This is ensured by the run termination condition of Table 6.1.2).

4): Cf. Figure 6.13. All these operations ensure that if S was located at some corner c_1 when the operation started, it afterwards either is located at some other corner c_2 such that both corners are rotated equally or terminates if the target corner has been removed during the operation (cf. Table 6.1.5, 4)). If not terminated, then, because still located on a quasi line (cf. 2)), again *a*) (OP-A) or *b*) (OP-B) can be applied or run passing is started.

The run passing needs a closer look, because it interrupts other operations. In order to ensure a regulated behavior, we chose the distance between sequent runs big enough such that a run does not have to execute a new run passing operation before it has finished its previous one. We settle the values for the constant L and the viewing path length appropriately. We look at two sequent runs S_1 and S_1^{succ} such that S_1^{succ} has been started after S_1. Their distance D is at least $L - 3$. This value is achieved if at the start of S_1 the operation *c*) (OP-C) and next, operation *b*) (OP-B) (Figure 6.13) was executed.

Now, we chose the value for the constant D big enough for ensuring that while some run S_2 is passing along S_1, S_1^{succ} becomes visible to S_2 the earliest when the passing operation with S_1 has been completed. Figure 6.16 shows an example of the longest possible duration of a run passing operation. Here, it takes 6 rounds until S_2 has arrived at its target corner. Because the run passing operation starts when the distance between S_1 and S_2 is ≤ 3, after the passing operation, the distance between S_2 and S_1^{succ} equals $D - 9$. We want this then to still be ≥ 3. So we choose $D \geq 12$ and together with the above argumentation concerning the minimum distance between sequent runs it follows $L \geq 15$. In

order to detect that the distance has become smaller than 12 ($= D$) (and solve this problem), the viewing path length must be 11 (cf. Table 6.1.2)).

5): When defining good pairs in Section 6.3.1, good pairs have been characterized by the relative position of the outer direct neighbors of the good pair according to the quasi line. The first part of the proof of 4) finishes the proof. □

CHAPTER 7

Conclusions & Outlook

We now close the thesis by giving a short summary of our achievements and envisioning future work in the topic of my research. For all our results, we used the fully synchronous $\mathcal{FSYNC}$ time model. First, we have formally shown that using an extremely simple and local robot model gathering of n robots on a grid can always be solved and provided an upper running time bound of $\mathcal{O}(n^2)$. The tightness of this bound w.r.t. our strategy was only informally shown by our suitable experiments. This also matches the formal lower bound that Degener et al. [Deg+11] showed for the Euclidean gathering strategy under the same robot and time model.

In the remaining part of the thesis, we extended the robot model by a constant-sized memory and a constant number of locally visible states. First, we formally proved an $\mathcal{O}(n)$ upper running time bound under this model, solving the same gathering problem as above. We further formally proved its asymptotic tightness.

Finally, we modified the connectivity and vision model to robots that are always connected as one initially given closed chain. There, we also formally proved an asymptotic optimal running time bound $\mathcal{O}(n)$, needed for the gathering.

Interestingly, the only minimal additional robot capabilities of a constant memory and constant number of locally visible states provided a huge im-

provement of the total running times.

Outlook For the future work the first step would be to develop a fast gathering strategy for the Euclidean plane that uses the robot model extension of constant-sized memory and a constant number of visible states. If one would also try to apply our concept of moving states (runs), this approach in principle could work.

Using this concept in the Euclidean plane, the main difficulty is to find starting positions for new runs: On the grid, we made use of the property that on the grid asymmetries of the swarm shape always exist (no circle possible). We used locally detectable asymmetries of the swarm shape for defining starting positions for new runs. In contrast, in the Euclidean plane, circular shaped swarms are possible, so that asymmetries cannot be used in order to settle run starting positions. One could instead exclude fully symmetric starting swarm configurations from valid inputs. But then the gathering process might still run into a circle configuration instead of a gathering point. So this may require some new concepts.

Assuming that for the Euclidean plane also an $\mathcal{O}(n)$ upper running time could be formally shown, then a formal proof of a general lower bound for the gathering under the extremely simple robot model that was used for our $\mathcal{O}(n^2)$ grid and for the $\mathcal{O}(n^2)$ Euclidean strategy is a very interesting result. If this general lower bound is actually $\Omega(n^2)$, then one would have shown that a very gentle extension (constant-sized memory and a constant number of locally visible states) of the robot model is the actual reason for a huge running time improvement.

Bibliography

[Abs+16] S. Abshoff, A. Cord-Landwehr, M. Fischer, D. Jung, and F. Meyer auf der Heide. "Gathering a Closed Chain of Robots on a Grid". In: *2016 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2016, Chicago, IL, USA, May 23-27, 2016*. IEEE Computer Society, 2016, pp. 689–699.

[AP06] N. Agmon and D. Peleg. "Fault-Tolerant Gathering Algorithms for Autonomous Mobile Robots". In: *SIAM Journal on Computing* 36.1 (2006), pp. 56–82.

[AQS14] M. K. Aguilera, L. Querzoni, and M. Shapiro, eds. *Principles of Distributed Systems - 18th International Conference, OPODIS 2014, Cortina d'Ampezzo, Italy, December 16-19, 2014. Proceedings*. Vol. 8878. Lecture Notes in Computer Science. Springer, 2014.

[ASY95] H. Ando, Y. Suzuki, and M. Yamashita. "Formation and agreement problems for synchronous mobile robots with limited visibility". In: *Proceedings of the 1995 IEEE International Symposium on Intelligent Control, ISIC 1995*. Aug. 1995, pp. 453–460.

[Bra+13] P. Brandes, B. Degener, B. Kempkes, and F. Meyer auf der Heide. "Energy-efficient strategies for building short chains of mobile robots locally". In: *Theoretical Computer Science* 509 (2013), pp. 97–112.

[Cie+03] M. Cieliebak, P. Flocchini, G. Prencipe, and N. Santoro. "Solving the Robots Gathering Problem". In: *Automata, Languages and Programming, 30th International Colloquium, ICALP 2003, Eindhoven, The Netherlands, June 30 - July 4, 2003. Proceedings*. Ed. by J. C. M. Baeten, J. K. Lenstra, J. Parrow, and G. J. Woeginger. Vol. 2719. Lecture Notes in Computer Science. Springer, 2003, pp. 1181–1196.

[CMN04] I. Chatzigiannakis, M. Markou, and S. E. Nikoletseas. "Distributed Circle Formation for Anonymous Oblivious Robots". In: *Experimental and Efficient Algorithms, Third International Workshop, WEA 2004, Angra dos Reis, Brazil, May 25-28, 2004, Proceedings*. Ed. by C. C. Ribeiro and S. L. Martins. Vol. 3059. Lecture Notes in Computer Science. Springer, 2004, pp. 159–174.

[COD07] A. L. Christensen, R. O'Grady, and M. Dorigo. "A Mechanism to Self-Assemble Patterns with Autonomous Robots". In: *Advances in Artificial Life, 9th European Conference, ECAL 2007, Lisbon, Portugal, September 10-14, 2007, Proceedings*. Ed. by F. A. e Costa, L. M. Rocha, E. Costa, I. Harvey, and A. Coutinho. Vol. 4648. Lecture Notes in Computer Science. Springer, 2007, pp. 716–725.

[Cor+11] A. Cord-Landwehr, B. Degener, M. Fischer, M. Hüllmann, B. Kempkes, A. Klaas, P. Kling, S. Kurras, M. Märtens, F. Meyer auf der Heide, C. Raupach, K. Swierkot, D. Warner, C. Weddemann, and D. Wonisch. "Collisionless Gathering of Robots with an Extent". In: *SOFSEM 2011: Theory and Practice of Computer Science - 37th Conference on Current Trends in Theory and Practice of Computer Science, Nový Smokovec, Slovakia, January 22-28, 2011. Proceedings*. Ed. by I. Cerná, T. Gyimóthy, J. Hromkovic, K. G. Jeffery, R. Královic, M. Vukolic, and S. Wolf. Vol. 6543. Lecture Notes in Computer Science. Springer, 2011, pp. 178–189.

[Cor+16] A. Cord-Landwehr, M. Fischer, D. Jung, and F. Meyer auf der Heide. "Asymptotically Optimal Gathering on a Grid". In: *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2016, Asilomar State Beach/Pacific Grove, CA, USA, July 11-13, 2016*. Ed. by C. Scheideler and S. Gilbert. ACM, 2016, pp. 301–312.

[CP04] R. Cohen and D. Peleg. "Robot Convergence via Center-of-Gravity Algorithms". In: *Structural Information and Communication Complexity, 11th International Colloquium , SIROCCO 2004, Smolenice Castle, Slowakia, June 21-23, 2004, Proceedings*. Ed. by R. Kralovic and O. Sýkora. Vol. 3104. Lecture Notes in Computer Science. Springer, 2004, pp. 79–88.

[DAn+12] G. D'Angelo, G. D. Stefano, R. Klasing, and A. Navarra. "Gathering of Robots on Anonymous Grids without Multiplicity Detection". In: *Structural Information and Communication Complexity - 19th International Colloquium, SIROCCO 2012, Reykjavik, Iceland, June 30-July 2, 2012, Revised Selected Papers*. Ed. by G. Even and M. M. Halldórs-

son. Vol. 7355. Lecture Notes in Computer Science. Springer, 2012, pp. 327–338.

[Das+12] S. Das, P. Flocchini, G. Prencipe, N. Santoro, and M. Yamashita. "The Power of Lights: Synchronizing Asynchronous Robots Using Visible Bits". In: *2012 IEEE 32nd International Conference on Distributed Computing Systems, Macau, China, June 18-21, 2012*. IEEE Computer Society, 2012, pp. 506–515.

[Das+16] S. Das, P. Flocchini, G. Prencipe, N. Santoro, and M. Yamashita. "Autonomous mobile robots with lights". In: *Theoretical Computer Science* 609 (2016), pp. 171–184.

[Déf+16] X. Défago, M. G. Potop-Butucaru, J. Clément, S. Messika, and P. R. Parvédy. "Fault and Byzantine Tolerant Self-stabilizing Mobile Robots Gathering - Feasibility Study -". In: *CoRR* abs/1602.05546 (2016). arXiv: 1602.05546.

[Deg+11] B. Degener, B. Kempkes, T. Langner, F. Meyer auf der Heide, P. Pietrzyk, and R. Wattenhofer. "A tight runtime bound for synchronous gathering of autonomous robots with limited visibility". In: *SPAA 2011: Proceedings of the 23rd Annual ACM Symposium on Parallelism in Algorithms and Architectures, San Jose, CA, USA, June 4-6, 2011 (Co-located with FCRC 2011)*. Ed. by R. Rajaraman and F. Meyer auf der Heide. ACM, 2011, pp. 139–148.

[Der+14] Z. Derakhshandeh, S. Dolev, R. Gmyr, A. W. Richa, C. Scheideler, and T. Strothmann. "Brief announcement: amoebot - a new model for programmable matter". In: *26th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2014, Prague, Czech Republic - June 23 - 25, 2014*. Ed. by G. E. Blelloch and P. Sanders. ACM, 2014, pp. 220–222.

[Der+15] Z. Derakhshandeh, R. Gmyr, A. W. Richa, C. Scheideler, and T. Strothmann. "An Algorithmic Framework for Shape Formation Problems in Self-Organizing Particle Systems". In: *Proceedings of the Second Annual International Conference on Nanoscale Computing and Communication, NANOCOM 2015, Boston, MA, USA, September 21-22, 2015*. Ed. by F. Fekri, S. Balasubramaniam, T. Melodia, A. Beirami, and A. Cabellos. ACM, 2015, 21:1–21:2.

[Der+16] Z. Derakhshandeh, R. Gmyr, A. W. Richa, C. Scheideler, and T. Strothmann. "Universal Shape Formation for Programmable Matter". In: *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2016, Asilomar State Beach/Pacific Grove, CA, USA, July 11-13, 2016*. Ed. by C. Scheideler and S. Gilbert. ACM, 2016, pp. 289–299.

[Des+06] A. Dessmark, P. Fraigniaud, D. R. Kowalski, and A. Pelc. "Deterministic Rendezvous in Graphs". In: *Algorithmica* 46.1 (2006), pp. 69–96.

[DK02] X. Défago and A. Konagaya. "Circle formation for oblivious anonymous mobile robots with no common sense of orientation". In: *Proceedings of the 2002 Workshop on Principles of Mobile Computing, POMC 2002, October 30-31, 2002, Toulouse, France*. ACM, 2002, pp. 97–104.

[DKM10] B. Degener, B. Kempkes, and F. Meyer auf der Heide. "A local $\mathcal{O}(n^2)$ gathering algorithm". In: *SPAA 2010: Proceedings of the 22nd Annual ACM Symposium on Parallelism in Algorithms and Architectures, Thira, Santorini, Greece, June 13-15, 2010*. Ed. by F. Meyer auf der Heide and C. A. Phillips. ACM, 2010, pp. 217–223.

[Dol+13] S. Dolev, R. Gmyr, A. W. Richa, and C. Scheideler. "Ameba-inspired Self-organizing Particle Systems". In: *CoRR* abs/1307.4259 (2013). arXiv: 1307.4259.

[DP12] Y. Dieudonné and F. Petit. "Self-stabilizing gathering with strong multiplicity detection". In: *Theoretical Computer Science* 428 (2012), pp. 47–57.

[Dyn+06] M. Dynia, J. Kutylowski, P. Lorek, and F. Meyer auf der Heide. "Maintaining Communication Between an Explorer and a Base Station". In: *Biologically Inspired Cooperative Computing, IFIP 19th World Computer Congress, TC 10: 1st IFIP International Conference on Biologically Inspired Computing, August 21-24, 2006, Santiago, Chile*. Ed. by Y. Pan, F. J. Rammig, H. Schmeck, and M. Solar. Vol. 216. IFIP. Springer, 2006, pp. 137–146.

[EP07] A. Efrima and D. Peleg. "Distributed Models and Algorithms for Mobile Robot Systems". In: *SOFSEM 2007: Theory and Practice of Computer Science, 33rd Conference on Current Trends in Theory and Practice of Computer Science, Harrachov, Czech Republic, January 20-26, 2007, Proceedings*. Ed. by J. van Leeuwen, G. F. Italiano, W. van der Hoek, C. Meinel, H. Sack, and F. Plasil. Vol. 4362. Lecture Notes in Computer Science. Springer, 2007, pp. 70–87.

[FJM17a] M. Fischer, D. Jung, and F. Meyer auf der Heide. "Gathering Anonymous, Oblivious Robots on a Grid". In: *Proceedings of the 13th International Symposium on Algorithms and Experiments for Wireless Sensor Networks, ALGOSENSORS 2017*. (accepted, to appear). 2017.

[FJM17b] M. Fischer, D. Jung, and F. Meyer auf der Heide. "Gathering Anonymous, Oblivious Robots on a Grid". In: *CoRR* abs/1702.03400 (2017). arXiv: 1702.03400.

[Flo+05] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. "Gathering of asynchronous robots with limited visibility". In: *Theoretical Computer Science* 337.1-3 (2005), pp. 147–168.

[Flo+14] P. Flocchini, G. Prencipe, N. Santoro, and G. Viglietta. "Distributed Computing by Mobile Robots: Solving the Uniform Circle Formation Problem". In: *Principles of Distributed Systems - 18th International Conference, OPODIS 2014, Cortina d'Ampezzo, Italy, December 16-19, 2014. Proceedings*. Ed. by M. K. Aguilera, L. Querzoni, and M. Shapiro. Vol. 8878. Lecture Notes in Computer Science. Springer, 2014, pp. 217–232.

[FPS12] P. Flocchini, G. Prencipe, and N. Santoro. *Distributed Computing by Oblivious Mobile Robots*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2012.

[Gra16] A. Graute. "Simulative Analyse eines Gathering Algorithmus für geschlossene Roboterketten auf einem Gitter". Bachelor's Thesis. Paderborn University, 2016.

[Izu+09] T. Izumi, T. Izumi, S. Kamei, and F. Ooshita. "Randomized Gathering of Mobile Robots with Local-Multiplicity Detection". In: *Stabilization, Safety, and Security of Distributed Systems, 11th International Symposium, SSS 2009, Lyon, France, November 3-6, 2009. Proceedings*. Ed. by R. Guerraoui and F. Petit. Vol. 5873. Lecture Notes in Computer Science. Springer, 2009, pp. 384–398.

[Izu+12] T. Izumi, S. Souissi, Y. Katayama, N. Inuzuka, X. Défago, K. Wada, and M. Yamashita. "The Gathering Problem for Two Oblivious Robots with Unreliable Compasses". In: *SIAM Journal on Computing* 41.1 (2012), pp. 26–46.

[Kat+07] Y. Katayama, Y. Tomida, H. Imazu, N. Inuzuka, and K. Wada. "Dynamic Compass Models and Gathering Algorithms for Autonomous Mobile Robots". In: *Structural Information and Communication Complexity, 14th International Colloquium, SIROCCO 2007, Castiglioncello, Italy, June 5-8, 2007, Proceedings*. Ed. by G. Prencipe and S. Zaks. Vol. 4474. Lecture Notes in Computer Science. Springer, 2007, pp. 274–288.

[KKR06] E. Kranakis, D. Krizanc, and S. Rajsbaum. "Mobile Agent Rendezvous: A Survey". In: *Structural Information and Communication Complexity, 13th International Colloquium, SIROCCO 2006, Chester, UK, July 2-5, 2006, Proceedings*. Ed. by P. Flocchini and L. Gasieniec. Vol. 4056. Lecture Notes in Computer Science. Springer, 2006, pp. 1–9.

[KM09] J. Kutylowski and F. Meyer auf der Heide. "Optimal strategies for maintaining a chain of relays between an explorer and a base camp". In: *Theoretical Computer Science* 410.36 (2009), pp. 3391–3405.

[KMP08] R. Klasing, E. Markou, and A. Pelc. "Gathering asynchronous oblivious mobile robots in a ring". In: *Theoretical Computer Science* 390.1 (2008), pp. 27–39.

[Li+17] S. Li, C. Markarian, F. Meyer auf der Heide, and P. Podlipyan. "A Continuous Strategy for Collisionless Gathering". In: *Proceedings of the 13th International Symposium on Algorithms and Experiments for Wireless Networks, ALGOSENSORS 2017*. (accepted, to appear). 2017.

[LM14] T. Lukovszki and F. Meyer auf der Heide. "Fast Collisionless Pattern Formation by Anonymous, Position-Aware Robots". In: *Principles of Distributed Systems - 18th International Conference, OPODIS 2014, Cortina d'Ampezzo, Italy, December 16-19, 2014. Proceedings*. Ed. by M. K. Aguilera, L. Querzoni, and M. Shapiro. Vol. 8878. Lecture Notes in Computer Science. Springer, 2014, pp. 248–262.

[LMP16] S. Li, F. Meyer auf der Heide, and P. Podlipyan. "The Impact of the Gabriel Subgraph of the Visibility Graph on the Gathering of Mobile Autonomous Robots". In: *Algorithms for Sensor Systems - 12th International Symposium on Algorithms and Experiments for Wireless Sensor Networks, ALGOSENSORS 2016, Aarhus, Denmark, August 25-26, 2016, Revised Selected Papers*. Ed. by M. Chrobak, A. F. Anta, L. Gasieniec, and R. Klasing. Vol. 10050. Lecture Notes in Computer Science. Springer, 2016, pp. 62–79.

[Lun+17] G. A. D. Luna, P. Flocchini, S. G. Chaudhuri, F. Poloni, N. Santoro, and G. Viglietta. "Mutual visibility by luminous robots without collisions". In: *Information and Computing* 254 (2017), pp. 392–418.

[Mar09] S. Martínez. "Practical multiagent rendezvous through modified circumcenter algorithms". In: *Automatica* 45.9 (2009), pp. 2010–2017.

[Mon+05] F. Mondada, L. M. Gambardella, D. Floreano, S. Nolfi, J. Deneubourg, and M. Dorigo. "The cooperation of swarm-bots: physical interactions in collective robotics". In: *IEEE Robotics & Automation Magazine* 12.2 (2005), pp. 21–28.

[Pel05] D. Peleg. "Distributed Coordination Algorithms for Mobile Robot Swarms: New Directions and Challenges". In: *Distributed Computing - IWDC 2005, 7th International Workshop, Kharagpur, India, December 27-30, 2005, Proceedings*. Ed. by A. Pal, A. D. Kshemkalyani,

R. Kumar, and A. Gupta. Vol. 3741. Lecture Notes in Computer Science. Springer, 2005, pp. 1–12.

[Pel12] A. Pelc. "Deterministic rendezvous in networks: A comprehensive survey". In: *Networks* 59.3 (2012), pp. 331–347.

[Pre07] G. Prencipe. "Impossibility of gathering by a set of autonomous mobile robots". In: *Theoretical Computer Science* 384.2-3 (2007), pp. 222–231.

[SG16] C. Scheideler and S. Gilbert, eds. *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2016, Asilomar State Beach/Pacific Grove, CA, USA, July 11-13, 2016*. ACM, 2016.

[Sha+17] G. Sharma, C. Busch, S. Mukhopadhyay, and C. Malveaux. "Tight Analysis of a Collisionless Robot Gathering Algorithm". In: *ACM Transactions on Autonomous and Adaptive Systems* 12.1 (2017), 3:1–3:20.

[SMM16] S. Saadatmand, D. Moazzami, and A. Moeini. "A Cellular Automaton Based Algorithm for Mobile Sensor Gathering". In: *Journal of Algorithms and Computation* 47.1 (2016), pp. 93–99.

[SN13] G. D. Stefano and A. Navarra. "Optimal Gathering of Oblivious Robots in Anonymous Graphs". In: *Structural Information and Communication Complexity - 20th International Colloquium, SIROCCO 2013, Ischia, Italy, July 1-3, 2013, Revised Selected Papers*. Ed. by T. Moscibroda and A. A. Rescigno. Vol. 8179. Lecture Notes in Computer Science. Springer, 2013, pp. 213–224.

[SN14] G. D. Stefano and A. Navarra. "Optimal Gathering on Infinite Grids". In: *Stabilization, Safety, and Security of Distributed Systems - 16th International Symposium, SSS 2014, Paderborn, Germany, September 28 - October 1, 2014. Proceedings*. Ed. by P. Felber and V. K. Garg. Vol. 8756. Lecture Notes in Computer Science. Springer, 2014, pp. 211–225.

[Sti16] M. Stienemeier. "Experimental Evaluation of a Gathering Strategy for Robots on a Grid". Bachelor's Thesis. Paderborn University, 2016.

[SY99] I. Suzuki and M. Yamashita. "Distributed Anonymous Mobile Robots: Formation of Geometric Patterns". In: *SIAM Journal on Computing* 28.4 (1999), pp. 1347–1363.