Algorithms for Mobile Agents with Limited Capabilities

Von der Carl-Friedrich-Gauß-Fakultät
Technische Universität Carolo-Wilhelmina zu Braunschweig

zur Erlangung des Grades
Doktor der Naturwissenschaften (Dr. rer. nat.)
genehmigte Dissertation
von
Dipl.-Math. Oec. Christiane Schmidt
geboren am 27.06.1981 in Wolfsburg.

Eingereicht am: 28.04.2011
Mündliche Prüfung am: 10.06.2011
Referent: Prof. Dr. Sándor P. Fekete
Korreferent: Prof. Dr. Joseph S. B. Mitchell

(2011)

Abstract

This work deals with geometric problems and the use of limited capability agents for these problems. Exploration and guarding problems have been extensively studied in computational geometry. The basic task is to monitor an environment (a polygon) either with a mobile guard or with a number of stationary guards (or guards with hard restrictions on the allowed movements). Though all these problems are easy to formulate—and have real-world interpretations that vividly illustrate them—some of these problems (as the classical art gallery problem) are hard to solve. The task of exploring an environment becomes more challenging if the environment is not known in advance, that is, there is no given ground plan and only areas that have already been visibly encountered by the explorer are known. Problems with this kind of uncertainty are called *online* problems. For *offline* problems the ground plan of the environment is known in advance. In this work we study one offline exploration problem and one exploration and guarding problem in the offline and online version, where the focus is on the latter. For our problems we do not deal with "almighty" explorers, but face limits on their perceptive capabilities. Other types of geometric problems can exploit this kind of limited agents: We present an exact and fast algorithm for an image analysis task with polyomino-shaped objects on a grid.

Distributed Vision with Smart Pixels. An important image analysis task is the identification of objects present in a given image. If these objects are to be processed automatically, not only the identification of the objects themselves, but also of certain attributes is of interest. The images we treat are pixel (grid) images, with a possibly huge number of intertwined objects. We make an assumption on the pixels: Besides light detection, they can perform simple computations and communicate with their grid neigh-

bors (*smart pixels*). Our goal is to extract attributes, such as the center of gravity or orientation, for each object in the image. In particular, we want to give a fast algorithm for this task. We show how the use of mobile agents, mimicked by messages sent by the pixels, allows for an exact algorithm—an agent sweep—that can cope with intertwined objects. We present how the attributes can be expressed as moments (of a random variable, considering the pixels as a point set in $\mathbb{R}^2$) and how the sweep accumulates the necessary information. For the algorithm we prove a runtime of only $O(W + H)$, with W and H being the width and height of the smallest bounding box for an object, respectively.

Exploration with a Myopic Watchman with Discrete Vision. In the classical *watchman route problem* the task is to find a shortest tour for an explorer such that each point of a given polygon is visible from at least one point of the tour. For this problem exact algorithms for a variety of polygon classes exist. We study this problem with two restrictions on the watchman's capabilities: The scan range is limited and visibility information can only be acquired at discrete points, "scanpoints". A scanpoint in combination with all points of the polygon that can be seen from the point and lie within its scan range form a scan. The scans must fully cover the given polygon. The cost for a tour of this watchman is a linear combination of tour length and number of scan points used along this tour. We show that this problem is NP-hard and present *approximation algorithms* for different variants: A 2.5-approximation for rectilinear grid polygons and unit L_∞ scan range, a 4-approximation for rectilinear grid polygons and unit L_2 scan range and a $\max(\frac{21}{4}, \frac{\pi r}{a} + \frac{\pi r}{2} + \frac{\pi}{2})$-approximation for the case of general polygons, an L_2 scan range and a bounded ratio r/a between visibility range r and minimum side length a. All these approximation algorithms can also be applied for the bicriteria version, that is, approximating the scan number and the tour length separately.

Exploration and Triangulation with a Swarm of Robots. The *art gallery problem* asks for a minimum number of (stationary) guards that allow for visibility coverage of a given polygon. Another classical problem in computational geometry is *triangulation*: The partition of a given polygon into triangles. We study a guarding problem linked to both of these problems: A swarm of agents with limited communication range has to establish a triangulated network in a given polygon. The edge lengths are limited to the communication range. The task is not only to give the positions, but to move to these locations in a connected fashion. Our goal is to minimize the number of robots used for this task, or, if the number of robots is limited, to cover as much area as possible with the triangulation established by these robots. We present NP-hardness results for both problems. Our focus is on

the online variants, we give a lower bound of 6/5 for the competitive ratio for any strategy for the first problem, as well as a 3-competitive strategy. We prove that the second problem does not allow for a constant competitive ratio.

Zusammenfassung

Die vorliegende Arbeit beschäftigt sich mit geometrischen Problemen und der Verwendung von Agenten mit eingeschränkten Fähigkeiten für diese Probleme. Explorations- und Überwachungsprobleme wurden in der algorithmischen Geometrie ausführlich betrachtet. Grundsätzlich besteht die Aufgabe in der Überwachung einer Umgebung (eines Polygons) mittels eines oder mehrerer mobiler oder stationärer Wächter (oder mittels mehrerer Agenten, die in ihren Bewegungsmöglichkeiten stark eingeschränkt sind). Einige dieser Probleme, wie das klassische Kunstgalerieproblem, können nur mit hohem Aufwand gelöst werden. Die Aufgabe, eine Umgebung zu explorieren, wird schwieriger, wenn die Umgebung nicht im Voraus bekannt ist, wenn also kein Grundriß gegeben und nur Bereiche, die bereits wahrgenommen wurden, bekannt sind. Probleme, die diese Art von Unsicherheit aufweisen, werden als *Online*-Probleme bezeichnet. Bei einem *Offline*-Problem ist der Grundriß der Umgebung von Beginn an gegeben. In dieser Arbeit betrachten wir ein Offline-Explorationsproblem, sowie die Offline- und Online-Variante eines Explorations- und Überwachungsproblems, wobei der Schwerpunkt auf dem Online-Aspekt liegt. Die von uns betrachteten Wächter haben nur beschränkte Wahrnehmungsfähigkeiten. Auch andere geometrische Probleme können sich solch beschränkter Agenten bedienen: für ein Problem der Bilderkennung mit polyominoförmigen Objekten in einem Gitter stellen wir einen exakten und schnellen Algorithmus vor.

Verteilte Sicht mit intelligenten Pixeln. Die Identifizierung von im Bild auftretenden Objekten ist eine wichtige Aufgabe im Rahmen der Bilderkennung. Für den Fall der automatischen Verarbeitung dieser Objekte ist nicht nur die Identifizierung der Objekte, sondern auch die Gewinnung bestimmter Attribute der Objekte von Interesse. Die von uns betrachteten

Bilder sind Pixelbilder, mit im Gitter angeordneten Pixeln und möglicherweise vielen, verwobenen Objekten. Wir treffen eine Annahme zu den Pixeln: über die Lichterkennung hinaus sind sie zu einfachen Berechnungen und zur Kommunikation mit ihren Gitternachbarn fähig. Daher sprechen wir von *intelligenten Pixeln*. Unser Ziel liegt in der Bestimmung bestimmter Objektattribute, wie dem Schwerpunkt und der Orientierung, für alle im Bild auftretenden Objekte. Insbesondere streben wir die Formulierung eines schnellen Algorithmus für diese Aufgabe an. Wir zeigen, wie ein solcher exakter Algorithmus unter Verwendung mobiler Agenten, implementiert durch das Versenden von Nachrichten durch die Pixel, erreicht werden kann. Dieser Sweepalgorithmus ist insbesondere in der Lage, verwobene Objekte zu behandeln. Wir zeigen, wie die Objektattribute mit Hilfe von Momenten (einer Zufallsvariablen) ausgedrückt werden können (unter Betrachtung der Pixel als Punktmenge im $\mathbb{R}^2$) und wie der Sweepalgorithmus die dazu notwendige Information sammelt. Wir beweisen eine Laufzeit von $O(W + H)$ für den vorgestellten Algorithmus, wobei W und H die Breite beziehungsweise Höhe der kleinsten Bounding Box bezeichnen.

Exploration mit einem kurzsichtigen Wächter und diskreter Sicht. Für das klassische *watchman-route*-Problem liegt die Aufgabe im Finden einer kürzesten Tour, so dass jeder Punkt des gegebenen Polygons von mindestem einem Punkt der Tour sichtbar ist. Für eine Reihe von Polygonklassen wurden für dieses Problem exakte Algorithmen angegeben. Wir betrachten einen Spezialfall mit zwei Einschränkungen der Fähigkeiten des Wächters: die Sichtweite ist beschränkt und nur an diskreten Punkten, „Scanpunkten", ist die Umgebungswahrnehmung möglich. Die an den diskreten Punkten aufgenommenen Scans müssen das gegebene Polygon vollständig überdecken. Damit ergibt sich auch eine Anpassung der Kostenfunktion für die Tour eines Wächters – die Kosten stellen sich als Linearkombination aus Tourlänge und Anzahl an Scanpunkten dar. Wir zeigen, dass dieses Problem NP-schwer ist und stellen *Approximationsalgorithmen* für verschiedene Varianten vor: eine 2.5-Approximation für orthogonale Gitterpolygone und einen Einheitssichtradius nach der L_∞-Norm, eine 4-Approximation für orthogonale Gitterpolygone und einen Einheitssichtradius nach der L_2-Norm sowie eine $\max(\frac{21}{4}, \frac{\pi r}{a} + \frac{\pi r}{2} + \frac{\pi}{2})$-Approximation für allgemeine Polygone, einen Sichtradius nach der L_2-Norm und einem beschränkten Verhältnis (r/a) von Sichtweite r und minimaler Seitenlänge a. Diese Approximationsalgorithmen können auch für die Variante des Problems, bei der Scananzahl und Tourlänge getrennt approximiert werden, angewandt werden.

Exploration und Triangulierung mit einem Roboterschwarm. Beim klassischen *Kunstgalerieproblem* soll die Anzahl der für die vollständige Überwachung notwendigen (stationären) Wächter minimiert werden. Ein weiteres

klassisches Problem der algorithmischen Geometrie ist die *Triangulierung* eines Polygons – dessen Zerlegung in Dreiecke. Wir stellen ein zu beiden Problemen in Zusammenhang stehendes Problem vor: ein Schwarm von Agenten mit beschränktem Kommunikationsradius soll ein trianguliertes Netzwerk in einem gegebenen Polygon aufbauen. Der eingeschränkte Kommunikationsradius impliziert eine Beschränkung der in der Triangulation auftretenden Kantenlängen auf dessen Wert. Die Aufgabe besteht nicht nur in der Bestimmung möglicher Positionen für die Agenten, sondern auch im Anfahren dieser Positionen unter Aufrechterhaltung des Zusammenhangs. Ziel ist es die Anzahl der für diese Aufgabe verwendeten Agenten zu minimieren, oder, falls nur eine beschränkte Anzahl von Robotern zur Verfügung steht, die durch die aufgebaute Triangulierung überdeckte Fläche zu maximieren. Wir zeigen, dass beide Probleme NP-schwer sind. Unser Schwerpunkt liegt auf den Online-Varianten der beiden Probleme. Wir geben eine untere Schranke von 6/5 für den kompetitiven Faktor jeder Online-Strategie und eine 3-kompetitive Strategie für das erste Problem an. Für das zweite Problem zeigen wir, dass keine deterministische Strategie einen konstanten kompetitiven Faktor erreichen kann.

Seht ihr den Mond dort stehen?
Er ist nur halb zu sehen
und ist doch rund und schön.
So sind wohl manche Sachen,
die wir getrost belachen,
weil unsre Augen sie nicht sehn.

Matthias Claudius, 1778

Acknowledgments

First of all, I would like to thank **Sándor Fekete**, my PhD advisor. I am very grateful for his support, encouragement and guidance and for giving me the opportunity to write this thesis. My special thanks go to **Joseph Mitchell** for his work on our joined papers that led to Chapters 4 and 5 and, in particular, for accepting to be the second referee of this thesis.

I feel very fortunate to have good friends. I would like to thank all of them for their help, support and time.

A big word of thanks goes to **Alexander Kröller**, for always taking his time for stimulating discussions, for many valuable thoughts and ideas—and for chocolate supplies, to **Nils Schweer** for providing feedback to my ideas and spending time for numerous talks and to **Tom Kamphans** for insightful discussions and helpful suggestions.

In particular, I am thankful to **Dörte Feichtenschlager** and **Martin Lorek** who proofread parts of this thesis.

Thanks to all my colleagues in the Algorithms Group for making the work much more fun.

Last but not least, I am very grateful to my parents, **Karin** and **Wolf-Dieter Schmidt**, who were always there for me, believed in me and supported me in every conceivable way.

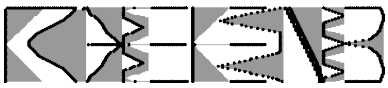
Contents

CHAPTER 1

Introduction

Geometry is a branch of mathematics that everybody knows from school. It is centuries old, with axioms already formulated by Euclid. Objects considered in geometry include points, lines, planes, circles, angles and distances. The original motivation of geometric research arose from "measuring the earth", which already indicates that besides the theoretical theorems the focus is on the real world with its strong geometric reference. If, for example, a person is located at home (position A) and wants to go to his office (position B), this person would in general be interested in a short(est) path between A and B. The positions can be described by coordinates in some coordinate system; if A and B are located in the open countryside, the distance may be measured according to the Euclidean metric, if the person is obliged to follow streets, the Manhattan metric may be used. Thus, we have a geometric embedding instead of two points whose positions do not provide any information on distances; the distance measure is based on this embedding. Another example arises from customer analysis: If we assume that a customer's decision for a certain shop is only influenced by the distance to the shop, a shop S wants to determine all customers that choose S. Again, this set of customers is given by the shop's and the customers' location in a geometrical embedded environment. To give a final example, think of W, a night watchman in a huge museum. He has to monitor all exhibits on his tour. The exhibits have a certain position in the museum and the positions from which an exhibit is visible depend on the location of obstacles like walls and doors.

What we can observe is not only this geometric reference, but there is often a certain question for which we might seek an answer: What is the

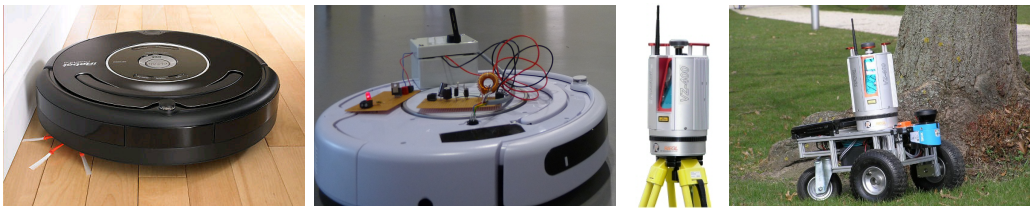


shortest path between A and B? What is the set of customers opting for S? What is a shortest tour for W, such that following this tour allows W to monitor all exhibits? In fact, we are not only interested in these answers for particular positions A and B, for a particular shop S or a particular museum. Instead, we want to give a solution for any such question (or at least for a great many of these questions). Hence, we want to give an algorithm for the problem.

A geometric context is often not present, and not necessary for the formulation of an algorithm. For example, we can formulate and solve a shortest path problem in a graph (e.g., using Dijkstra's algorithm, see [Sch03]). The edge lengths in this graph need not be related to the positions of the vertices, i.e., we may impose an arbitrary distance measure.

However, geometry often helps to give an algorithm. In many scenarios additional information provided by geometry is essential for a solution, or at least allows for the design of faster algorithms. For example, the "customer problem" presented above can be solved for shops in the plane: The Voronoi diagram of the set of shops can be determined (see, e.g., Aurenhammer and Klein [AK00]). These kinds of problems are the subject of computational geometry.

Two classical problems in computational geometry are exploration and guarding problems. The underlying task is to monitor an environment (a polygon) with one mobile guard W or with a number of stationary guards (or guards with hard restrictions on the allowed movements). In robot motion planning, *exploration* refers to the task of finding a path such that every point of the environment becomes visible from at least one point along the way. Other popular problems in this context are *navigation*, the task of finding a path to a target with known location, *search*, the task of finding a target with unknown location, and *localization*, the task of identifying ones own position in a known environment map. Over the years, these problems have been studied for varying environments and robots with different capabilities; other variants arise if a possibly huge number of robots is used rather than a single robot. An example is the *watchman route problem* stated by Chin and Ntafos [CN86], which asks for a shortest route for a single explorer along which every point of the environment becomes visible. The challenge is harder if the given polygon features holes—one may think of columns in a hall. In fact, the problem is computationally hard, i.e., only enormous computational effort can lead to an optimal solution for the problem. A classical guarding problem is the *art gallery problem*, stated by Victor Klee: A minimum number of (stationary) guards is to be positioned, such that complete visibility coverage of the environment is achieved.



(a) (b) (c) (d)

Figure 1.1: (a) An iRobot Roomba (image source: `www.irobot.com`), (b) an iRobot Roomba equipped with a sensor node, (c) the 3D Terrestrial Laser Scanner RIEGL VZ-400 (image source: `www.riegl.com`) and (d) the robot Irma3D (image source: `http://www.faculty.jacobs-university.de/anuechter/robots.html`).

In a classical algorithmic approach all information is assumed to be given. However, in many important scenarios we face a variety of restrictions. The agents may have limited capabilities, or the information about the environment may not be available in advance (resulting in an *online problem*). In addition, the explorer may have locally available information only; hence, *distributed algorithms* try to solve a problem based on local computations and communication.

Restricting the agents' capabilities may result in agents that are closer to "real" explorers. For example, the general statement of the watchman route problem assumes that the explorer has continuous 360° vision and can perceive objects at arbitrarily large distances—assumptions that hardly any human or automatic explorer will match.

In recent years, technologies for autonomous robots improved tremendously. Amongst other things this development resulted in commercially available platforms. By now, some of these platforms are even designed for basic everyday tasks such as vacuum cleaning, as is the iRobot Roomba shown in Figure 1.1(a). The price of these robots has been decreasing and the robots have gained popularity. The improved capabilities and comparatively low price makes their use for a great many tasks appealing. Still, they come with limited capabilities compared to the "ideal" agent for these tasks, leaving the agents with only local information.

In particular, swarms of robots become affordable. On the other hand, using a swarm of simple robots, as the iRobot Roomba, requires cooperation. The robots may easily be equipped with wireless sensor nodes (see Figure 1.1(b)), enabling them to communicate with each other, but only within a limited communication range.

Moreover, using commercially available platforms in the "as-is-state" after delivery for some kind of perception task restricts the sensing range; sensing

is restrained to contact-sensing bumpers and the like.

Not only the capabilities of commodity robots have improved. Also the development of sophisticated mobile robots has advanced in recent years. For example, the use of up-to-date 3D laser scanners, such as the scanner shown in Figure 1.1(c), enhanced the perception of the environment of autonomous robots; see Figure 1.1(d). This progress literally predestines those robots for exploration tasks: A robot equipped with a 3D laser scanner allows for sensing and—as it is mobile—it may be used for the exploration of an environment. However, amongst other things, the operation of a laser scanner limits the range of visibility—in comparison to an ideal agent with unlimited visibility range. In addition, properly joining the single scans often asks for a stationary scanner.

So, technical and price development allows for the use of these limited capability devices. But, we have to deal with new algorithmic questions.

One approach is to consider the minimal necessary capabilities that allow for a certain task. Suri et al. [SVW08] and Brunner et al. [BMS+08] classified different robot models along these lines.

Here, we focus on using these agents for concrete geometric problems. Obviously, an agent used for an exploration task needs to be mobile in order to roam the polygon. We are interested in exploration strategies for a robot with limited and discrete vision, like the robot equipped with a laser scanner described above: the robot has a limited range of visibility and needs to be immobile while taking a scan of the (polygonal) environment. Both moving and scanning take time. Hence, when we ask for a shortest tour, these time requirements should be reflected in the cost. That is, the cost function is a linear combination of the tour length and the number of scans. This kind of natural time measurement has not yet been considered.

Moreover, we consider the use of mobile agents—a swarm—for a guarding task: agents with a limited communication range, as the Roombas equipped with a sensor node described above, should be used to extend a given sensor network into a polygon. The moderate price of these commodity robots allows for their integration for this task. The goal is an extended network that preserves communication lines among different agents. The agents have to head for the positions in the environment. In particular, we ask for a certain structure—a triangulated network: such a network allows for localization of entities moving in the environment. Of course, we would like to cover the entire region with the triangulated structure, or, in case we face a limit on agents we are allowed to use, we would like to cover as much area as possible.

The surprising observation we can make is that this kind of mobile limited capability agents can be of use also for other geometric problems. So far, we

considered a single agent and a swarm of agents for which this restriction was intrinsic and sought algorithms for the resulting problems. Using geometric information and limited agents can also lead to better (i.e., faster) algorithms for problems which at first sight actually only feature the geometric reference.

A video camera that records a picture uses a camera chip. These chips are composed of sensor pixels, in general arranged in a rectangular grid. The geometry of input objects—in particular if the picture is taken of objects on a planar background—and the geometry of processors that capture these objects is strongly correlated. Hence, for image analysis tasks we may exploit this given geometry.

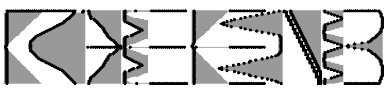
Technical development of camera chips allows for more than this geometric match. "Smart" sensor pixels are equipped with a limited CPU and limited local memory. Hence, instead of being restricted to the pure task of light detection (and forwarding the obtained information), these pixels can perform limited computations, communicate within a limited range and work with a limited storage.

This kind of processor element grids has been used for a variety of non-geometric problems. For example, Thompson and Kung [TK77] presented an algorithm for sorting n^2 numbers using a $n \times n$ grid of processor elements. Higher level image processing task have been considered by Miller and Stout [MS85].

As described above, for image analysis tasks we can use the geometry. Moreover, using such a smart pixel grid, we can simulate moving agents with these pixels—by sending messages. Thus, those agents are limited in computation and communication. We make use of these agents for a special image analysis task. These agents are not intrinsic to the problem. We aim at extracting information, such as an object's orientation, for all objects (black polyominoes) in a given black-and-white pixel image. The pixels of the camera chip are "smart" as described above, and we use agents simulated by these pixels for determining the desired information. The limited capability agents allow for an algorithm that is faster than any centralized algorithm.

Thus, in this work we consider three geometric problems in 2-dimensional space, each dealing with polygons or polyominoes. Moreover, the algorithms we present make use of mobile agents with limited capabilites. For two problems these agents are intrinsic, for one problem their use allows for a fast algorithm.

Outline of this Thesis. In Chapter 2 we give basic definitions for the rest of this work.



Chapter 3 presents an algorithm for the problem of computing the size, the center of gravity and the orientation of objects in a given black-and-white pixel image. We describe how these attributes can be computed, in particular, using a Principal Component Analysis, give a (distributed) agents' sweep over the objects that allows for the computation and analyze the algorithm's running time.

Chapter 4 studies a variant of the classical watchman route problem: the *myopic watchman problem with discrete vision*. The agent exploring the polygon has a limited range of visibility and is restricted to taking scans at discrete points. In the classical problem we ask for a shortest tour, and this refers to the tour length only. Of course, for the new problem we have to adjust the objective function: The cost is a linear combination of tour length and number of scans. Since both moving and scanning take time, this reflects the search for a "shortest" tour in terms of time spent for the entire coverage. Since we prove this problem to be NP-hard, our focus is on approximation algorithms. For different variants of the problem, varying the class of polygons and the norm for the scan range, we present approximation algorithms.

The problem of exploring a polygon and establishing a triangulated network with agents with limited communication range is studied in Chapter 5. We aim at using a minimum number of agents for this task for the entire region, or at maximizing the covered region if we face a hard limit on the number of agents. We present complexity results. As the focus of interest is the expansion of a network into a new, unknown area, we study the online versions of the problem and give a lower bound on the competitive ratio for any strategy and a competitive strategy for the first problem. For the second problem we show that no competitive strategy exists.

Each chapter begins with a problem statement, necessary definitions for the chapter and work related to the problem.

The work presented in this thesis was prepared in collaboration with other people. The paper [FFK+09] (and a predecessor paper that did not contain the distributed sweep [KKS+08]) forms the basis for Chapter 3. It was prepared in collaboration with Sándor P. Fekete, Alexander Kröller, Dietmar Fey, Marcus Komann, and Marc Reichenbach. Chapter 4 is based on the paper [FMS10] (with a preliminary version [FMS09]). It was joint work with Sándor P. Fekete and Joseph S.B. Mitchell. Chapter 5 is based on work together with Sándor P. Fekete, Tom Kamphans, Alexander Kröller, and Joseph S.B. Mitchell ([FKK+11]).

CHAPTER 2

Preliminaries

In this chapter we introduce basic concepts that are needed in the following chapters. We do not give all terms in detail but explain the general concepts. For rigorous definitions and more details we refer to textbooks for the different topics. Definitions that we use only in a particular chapter, are given in the beginning of that chapter.

Mathematical Basics. We let $\mathbb{N}$ denote the set of positive integers and $\mathbb{Z}$ denote the set of all integers. The set of all rational numbers is denoted by $\mathbb{Q}$ and the set of all reals by $\mathbb{R}$. The n-dimensional vector space over the reals is denoted by $\mathbb{R}^n$. For a n-dimensional vector x the L_2-norm (Euclidean norm) is defined as $\|x\|_2 := (\sum_{i=1}^{n} x_i^2)^{1/2}$, the L_1-norm (Manhattan norm) as $\|x\|_1 := \sum_{i=1}^{n} |x_i|$, and the L_∞-norm as $\|x\|_\infty := \max_{i=1,\ldots,n} |x_i|$.

Algorithms and Computational Complexity. A *problem* Π is a general question, for which we seek an answer. It consists of a set of parameters and a description of desired properties of the answer, the solution. An *instance* $I \in \Pi$ of a problem defines values for all problem parameters. A *decision problem* is a problem to which the answer is either "yes" or "no". For an *optimization problem* an instance is given by a solution set $\mathcal{S}$ and an objective function $f : \mathcal{S} \to \mathbb{R}$. The answer is either the optimal value of the objective function or, in case $\mathcal{S} = \emptyset$, "infeasible", or "unbounded". For a *maximization problem* the optimal value of the objective function, $|\mathrm{OPT}|$, is $\max_{x \in \mathcal{S}} f(x)$, for a *minimization problem* $\min_{x \in \mathcal{S}} f(x)$. Each problem's description implicitly defines an input alphabet and an encoding scheme (that maps problem instances to a string of symbols from this alphabet). The *input size* of an

instance $I \in \Pi$ is defined as the number of symbols of the underlying alphabet that are necessary to encode I. In general, *binary encoding*, that is, only ones and zeros, is used. An *algorithm* for a problem Π is a precise sequence of instructions that can be applied to any instance $I \in \Pi$ and that guarantees to end with a solution for I. We say that an algorithm solves a problem in *polynomial time* if we can bound its running time for any $I \in \Pi$ by a polynomial function of I's input size. A decision problem that allows for a polynomial time algorithm belongs to the class P. If the running time of an algorithm can be bounded by a polynomial function of the input size and the maximum input value, we say that the algorithm is *pseudopolynomial*. A problem belongs to the class NP, if for every "yes"-instance there is a certificate that can be checked in polynomial time. That is, a given solution can be verified in polynomial time. Obviously, P$\subseteq$NP. We say that a problem Π is *NP-hard* if $\Pi \in$P also proves P=NP. It is widely believed that P$\neq$ NP. Under this assumption NP-hardness of a problem implies that no polynomial algorithm exists for this problem. *NP-complete* problems are NP-hard problems that are also in NP. A polynomial time transformation of a problem $\Pi' \in$ NP into an instance of problem Π is called a *reduction*. Such a transformation T produces for every input x of Π' an input $T(x)$ of Π such that the answer—"yes" or "no"—for $T(x)$ as an input for Π is the correct answer for x as an input for Π'. Given such a reduction: If there is a polynomial time algorithm for Π, then there is also a polynomial time algorithm for Π'. Thus, giving such a reduction from an NP-complete problem Π' proves Π to be NP-hard.

Decision problems can be derived from optimization problems. For example, if we are given a maximization problem, that is, a problem that asks for a solution with some maximum "profit" we may also add a numerical bound B as a parameter and ask whether there is a solution resulting in a profit of at least B. Hence, all the terms presented above are commonly used also for optimization problems.

For further reading we recommend the textbooks by Papadimitriou [Pap94] and Garey and Johnson [GJ90]. Garey and Johnson provide a detailed list of NP-complete problems.

In particular for NP-hard problems we are interested in polynomial time *approximation algorithms*. These do not necessarily produce an optimal solution for every instance of a problem, but guarantee to give a solution that is at most a constant times the optimal solution (for a minimization problem). So, if we consider a minimization problem, we want to bound |ALG|, the objective function value computed by the algorithm ALG, in the form

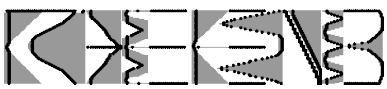
$$|\mathrm{ALG}| \leq \alpha \cdot |\mathrm{OPT}| + \beta \tag{2.1}$$

for constants α and β (α is also allowed to depend on the input size). An al-

gorithm that allows for such a bound is called an *asymptotic α-approximation algorithm* (or *asymptotic α-approximation*). For $\beta = 0$ ALG is called an *absolute α-approximation*, we then refer to α as the *approximation factor* (or *approximation ratio*). For a minimization problem $\alpha > 1$, and we attempt to obtain an approximation algorithm with an α as close to 1 as possible. For a maximization problem the $\leq$ is substituted for a $\geq$ in Equation 2.1. In this case $\alpha < 1$ and again we are interested in an approximation algorithm with an α close to 1. A *polynomial time approximation scheme* (PTAS) is an algorithm that provides an $(1 + \varepsilon)$-approximation for any problem instance and any $\varepsilon > 0$ (for a minimization problem). More precisely, a PTAS gives an approximation algorithm for any fixed ε. Such an approximation scheme is called a *fully polynomial time approximation scheme* (FPTAS) if the running time can be bounded by a polynomial function of the input size and $1/\varepsilon$. Textbooks on approximation algorithms are provided by Hochbaum [Hoc97] and Vazirani [Vaz01].

For the analysis of an algorithm's runtime, we use *Landau symbols*. Given two functions $f, g : \mathbb{N} \to \mathbb{R}$, we say f is $O(g(n))$ (and denote it by $f(n) = O(g(n))$ or $f = O(g)$) if: $\exists c > 0, n_0 \in \mathbb{N}, \forall n \geq n_0 : 0 \leq f(n) \leq c \cdot g(n)$. So, we use this notation to give an upper bound on f's running time. If an algorithm's input is from a set X and we are given a function $f : X \to \mathbb{R}$, such that the algorithm terminates after at most $\alpha f(x)$ elementary steps (for every possible input $x \in X$) we say that the algorithm *runs* in $O(f)$ time. The *running time* of the algorithm is said to be $O(f)$. The definition of a polynomial time algorithm can be given in terms of this notation: In case $f(n) = O(n^k)$ for a constant k, we call f *polynomially bounded*. An algorithm for which the running time is polynomially bounded runs in *polynomial time*. For $k = 1$ the algorithm is called a *linear-time algorithm*. A lower bound on the running time is denoted in the Ω-notation. We say f is $\Omega(g(n))$ (and denote it by $f(n) = \Omega(g(n))$ or $f = \Omega(g)$) if: $\exists c > 0, n_0 \in \mathbb{N}, \forall n \geq n_0 : f(n) \geq c \cdot g(n) \geq 0$. In case there exists a $q > 0$ such that $f(n) = \Omega(q^n)$, we say that f grows at least *exponentially*. A function f with $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$ is denoted by $f(n) = \Theta(g(n))$. Hence, this denotes that f and g have the same rate of growth.

The algorithmic notation we gave so far refers to an offline situation. That is, the entire information is available from the start. An algorithm for an online scenario with incomplete information (on future input) is analyzed similar to an approximation algorithm. Again, we consider a minimization problem. Here, the objective function value computed by the *online algorithm* ALG, |ALG|, is compared to the value of an optimal offline solution, |OPT|. The online algorithm is called a *c-competitive algorithm* if we can



bound |ALG| in the form:

$$|\text{ALG}| \leq c \cdot |\text{OPT}| + \beta \tag{2.2}$$

for a constant β. We may refer to the algorithm as a *strictly c-competitive algorithm* if $\beta \leq 0$. c is called the *competitive ratio* or *competitive factor* of the algorithm; it may be a constant or depend on the input size (but not on the input). The competitive factor is larger than 1; we desire online algorithms with a ratio as close to 1 as possible. Again, for a maximization problem we modify Equation 2.2 by exchanging the "less than or equal to" for a "greater than or equal to". We recommend the textbooks by Fiat and Woeginger [FW98] and Borodin and El-Yaniv [BEY98] for more details on online algorithms.

Polygons and Graphs. A *polygon* P is given by a set of n *vertices* $v_1, v_2, \ldots, v_n$ and n *edges* $v_1v_2, v_2v_3, \ldots v_{n-1}v_n, v_nv_1$ such that v_iv_j and v_kv_l for $j \neq k$ do not share a point. The polygon is the closed finite region in the plane bounded by the vertices and edges such that there exists a path between any two points of P which does not intersect any edge. The edges and vertices form the *boundary* of P. The boundary is denoted by ∂P (with $\partial P \subseteq P$). Since the polygon is closed and bounded, the boundary consists of cycles of edges. Two consecutive edges in a cycle share an edge. In case the boundary consists of two or more cycles, P is a polygon with holes. If the polygon has no holes, it is called a *simple polygon*.

A *graph* G is a triple (V, E, Ψ), where V denotes the finite set of *vertices* (or *nodes*) and E denotes the finite set of *edges*. For an undirected graph $\Psi : E \to \{X \subseteq V : |X| = 2\}$ and an edge is denoted as $e = \{v, w\}$. For a directed graph (or digraph) $\Psi : E \to \{(v, w) \in V \times V : v \neq w\}$ and an edge is denote as $e = (v, w)$. Often, only $G = (V, E)$ is used. Two vertices are said to be *adjacent* if they are joined by an edge. A sequence $v_1, e_1, v_2, \ldots, v_{k-1}, e_{k-1}, v_k$ with $e_i \neq e_j$ for $1 \leq i < j < k$ is a *walk* in G, which is closed if $v_1 = v_k$. A closed walk with $v_i \neq v_j$ for $1 \leq i < j < k$ is a *cycle*. When a connected graph does not contain a cycle as a subgraph it is called a *tree*. A *path* is a walk that visits no vertex twice.

CHAPTER 3

Distributed Vision with Smart Pixels

In this chapter we study the problem of extracting information, such as the center of gravity, for objects given in a black-and-white pixel image. Instead of using a single central processor we assume that the pixels of the grid are "smart": The pixels may communicate with their immediate neighbors (exchanging low-level information) and perform simple computations.

3.1 Problem Description

Given a $W \times H$ black-and-white pixel image depicting several objects, we would like to distinguish these objects and compute several attributes for each object: its size, its center of gravity, and its orientation. Using a classical algorithmic approach we could use a single central processor with access to the information of all pixels. With this global information the processor is then able to determine the desired attributes.

When looking at the real-life application that provoked our interest in this problem, one is immediately convinced that the image processing described above has to be extremely fast. A video camera at a robot arm, as shown in Figure 3.1, should enable the robot to identify and seize machine pieces moving on a conveyor belt. In order to seize the objects the robot must compute the object's size, a point where it should grasp the object, e.g. the center of gravity, and the orientation in space of all objects in the current image. The machine pieces on the conveyor belt rapidly leave the robot's range of operation. Thus, actually seizing the objects requires a fast computation.

Figure 3.1: Robot arms at a conveyor belt. image source: `coriolinus @ www.flickr.com`.

On the other hand, there is a special hardware that can be used; see Figure 3.2. In general, a camera chip is composed of sensor pixels arranged in a rectangular grid. Each of the sensor pixels can detect light. While a standard pixel is limited to forwarding the data resulting from this perception to a central unit, a *smart pixel* is able to process the pixel info, communicate with its direct neighbors, to perform simple computations (such as addition and subtraction), and to react to events. This hardware is not just a theoretical concept, but actually does exist [FHL+04]. Instead of a single processor we are confronted with many parallel processors, limited to local communication. This allows for a significant speedup. Another crucial observation can be made: The geometry of the input and the geometry of the processor locations are strongly correlated. The same holds for the size of the input and the number of processors involved. Thus, relying on the processor grid rather than on a single processing unit for the basic image processing, we may achieve a faster computation.

For the concrete algorithm: Each sensor pixel is a processor, limited to simple computations and communication with its neighbors. With those pixels we simulate agents sweeping over the pixel image. Hence, those agents are also limited in computation and communication. The decisions are made based on local information only. The correlation between input and processor location allows for a fast sweep that provides the desired attributes.

To clarify this, consider an object of $\Theta(WH)$ pixels and with a diameter in $\Theta(WH)$, as shown in Figure 3.3. There is a lower bound of $\Omega(WH)$ on the runtime for any centralized algorithm on such an object. In contrast, we

will present a distributed algorithm with runtime $O(W + H)$. Still, the final computations and handling for whatever activity the machine was built, can be performed by a centralized processor.

3.1.1 Problem Statement

We observe objects on one conveyor belt and can restrict ourselves to a bichromatic image—enabling us to distinguish the objects from the belt. We are given a black-and-white photo $\mathcal{X}$ overlayed on a pixel grid: Object pixels are black, white pixels are part of the background. The size of this pixel image is $W \times H$. $\mathcal{X}$ may consist of multiple objects; see Figure 3.4. We assume that the image is embedded in a grid of infinite size, such that we do not have to deal with special cases in the notation we give below.

Pixels of the infinite grid do not have a unique identifier, that is, neither an ID nor coordinates. Thus, the pixels are identical—apart from the

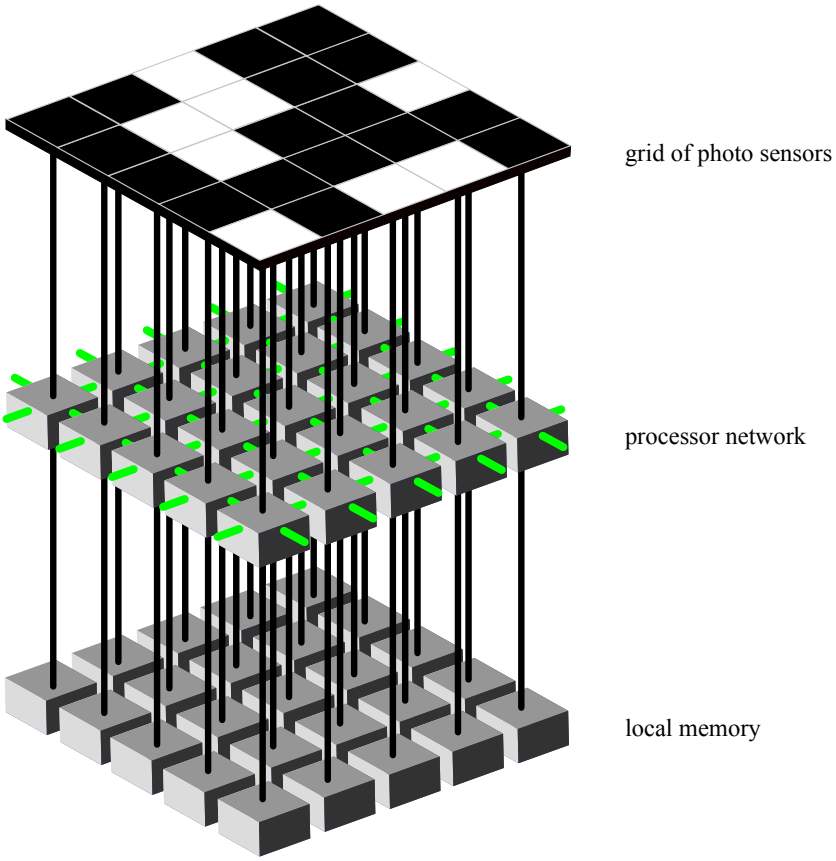


Figure 3.2: Schematic representation of the used hardware.

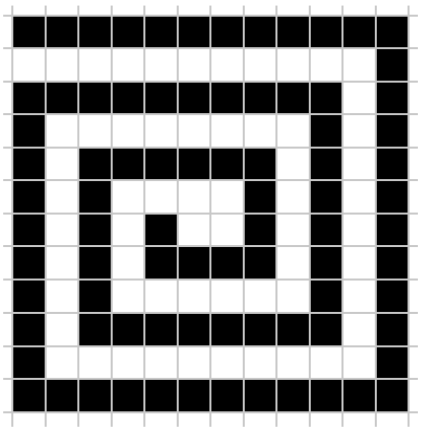


Figure 3.3: An object with $\Theta(WH)$ pixels and with a diameter in $\Theta(WH)$. Example for the lower bound of $\Omega(WH)$ for any centralized algorithm and for any distributed algorithm that stays within the object.

color. With this assumption we may reset the pixel grid to an initial state between consecutive runs without the need for a global initialization algorithm. All pixels are initialized with the same state. Each pixel p is able to communicate with its eight direct neighbors in L_∞-distance 1. We denote this set of neighbors by $\Gamma(p)$ and name the pixels in $\Gamma(p)$ by direction with respect to p: $\mathsf{N}(p)$, $\mathsf{NW}(p)$, $\mathsf{W}(p)$, $\mathsf{SW}(p)$, $\mathsf{S}(p)$, $\mathsf{SE}(p)$, $\mathsf{E}(p)$, and $\mathsf{NE}(p)$, with interpretation according to the cardinal points (of the compass). We define $\overline{\Gamma}(p) := \Gamma(p) \cup \{p\}$.

We can use moving agents on our pixel grid. The agents have only local behavior and knowledge. A moving agent is reflected by sending messages from one pixel to a neighbor.

Reflecting the limited CPU and small capacity associated with a single pixel, we only allow for the performance of simple computations (i.e., for

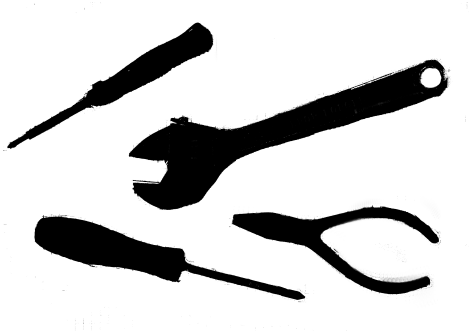


Figure 3.4: A black-and-white photo $\mathcal{X}$ with 4 objects.

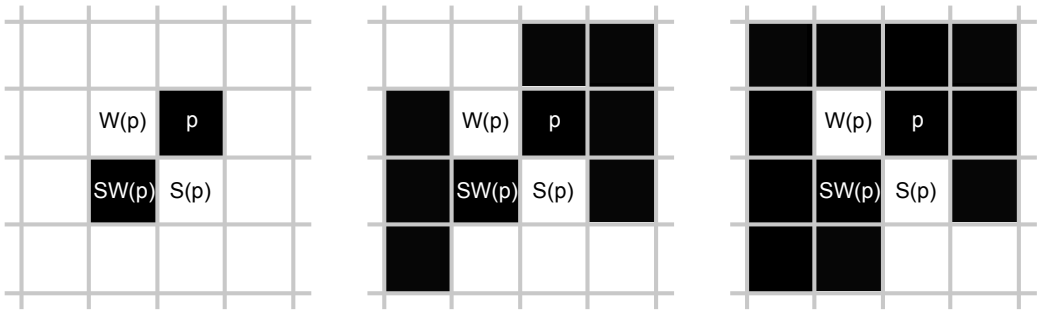


Figure 3.5: Two black pixels p and $\mathsf{SW}(p)$ (a). If $\mathsf{S}(p)$ and $\mathsf{W}(p)$ are white, they may belong to the same object (b), but may not (c).

example addition and subtraction). Thus, in addition, the agents have only limited computational power.

We assume the $\mathcal{LOCAL}$ model of Peleg [Pel00] for the communication between pixels. Thus, the network (of pixels) runs in synchronized rounds, with all pixels starting in the same round. Moreover, the local computation is unlimited, i.e., each pixel may perform any computation based on the data on hand. Also, the messages sent by the pixels of the grid may have unlimited size. Some more assumptions are made for the models of Peleg: All processors, that is, pixels, in our pixel grid are fault free. No pixels will be added or removed from the grid during the run of the algorithm. Our grid has infinite size, so no pixels can be added.

The black pixels define a set of objects in $\mathcal{X}$; an object consists of a connected set of black pixels. We define connectivity horizontally and vertically only. So, two pixels p_1 and p_2 in $\mathcal{X}$ belong to the same object if we can connect p_2 to p_1 by a path in $\mathcal{X}$ (i.e., a path using only black pixels), such that all pixels on this path are either vertically or horizontally adjacent to the next pixel on the path. In terms of the notation from above: Two black pixels p and $\mathsf{SW}(p)$ may or may not be part of the same object if $\mathsf{S}(p)$ and $\mathsf{W}(p)$ are white; see Figure 3.5. Note that the communication graph and the connectivity graph differ: Communication is possible with eight direct L_∞-neighbors, whereas for connectivity only the neighbors in L_1-distance 1 are considered.

Task. Based on the black-and-white pixel image the different objects (the connected components of the 4-regular grid graph with the connectivity definition from above) are to be distinguished. For each object certain attributes should be computed: the object's size (i.e., number of black pixels occupied by the object), its center of gravity, and its orientation. Here, we define the orientation to be the direction of maximum variance.

Organization of this Chapter. The remainder of the current section presents related work. In Section 3.2 we describe how the desired attributes can be deduced from the moments of the point set of object pixels and how relative moments are computed. We present our algorithm in Section 3.3. In the subsequent Section 3.4 we prove the algorithm's correctness and analyze its runtime. Moreover, we analyze the number of agents per pixel and the algorithm's applicability. We conclude in Section 3.5.

We published the results presented in this chapter in [FFK+09] (with a predecessor paper [KKS+08] which presented the distributed mechanisms to compute the desired values, but did not contain the distributed sweep algorithm).

3.1.2 Related Work

Mesh-Connected Computers. *Mesh-connected computers* (MCCs) have the same layout as the pixel grid we consider and have been considered for a long time. An MCC is a single instruction stream – multiple data stream (SIMD) computer. An MCC consists of processing elements (PEs) arranged in a $n \times n$ grid (n^2 PEs). In general, the assumption is made that n is a power of two. Each PE may communicate with its 4 neighbors (to the north, east, south and west). For our model we assume communication with 8 neighbors. This does not significantly change the setting when we measure in O-notation only. Each PE has (a constant number of) registers of size $\Theta(\log(n))$ and can perform standard arithmetic operations (on the content of these registers) in constant time. Sending or receiving messages from neighboring PEs is possible in $\Theta(1)$ time. See Miller and Stout [MS85] for definitions. PEs are referred to by row and column: PE(i, j) is located in row i and column j.

MCCs have been used to solve a variety of non-geometric problems. For example, graph problems, like finding the bridges and articulation points of an undirected graph or a minimum spanning tree, have been considered by Atallah and Kosaraju [AK84]. Nassimi and Sahni [NS80] gave an algorithm for computing the connected components of a graph with limited degree.

Many of these results use standard MCC algorithms, for example, the implementation of a transitive closure algorithm by Van Scoy [VS80] that allows to solve recurrences of the form

$$f_k(i,j) = g(f_{k-1}(i,j), f_{k-1}(i,k), f_{k-2}(k,j)) \qquad \forall\, f_n(i,j)$$

in time $\Theta(n)$ if g can be computed in $O(1)$ by a single PE and $f_0(i,j)$ is initially stored in PE(i,j). Thompson and Kung [TK77] presented an algorithm

for sorting n^2 numbers in $O(n)$ steps. Other standard algorithms include rotating data within a row or column, random access read and random access write; see Miller and Stout [MS85] for an overview.

In addition, MCCs have been used for image processing. That is, a picture of n^2 pixels is distributed on the MCC: one pixel per PE. Again, our model is equivalent to the MCC model for these problems. So, the geometry of the input is strongly correlated to the geometry of the processor location. Consequently, local operations on the image, such as edge detection, can be executed by local operations on the MCC. In addition to these low-level image processing tasks Miller and Stout [MS85] proposed the use of MCCs for higher level image processing tasks. They presented $\Theta(n)$ time algorithms for, amongst others, computing the extreme points of a convex hull, for deciding whether a component is convex and for determining internal diameters of components without holes. For their computation they use matrices, repeatedly rotate data, use the generalized transitive closure operation and sort data. They describe the sort-like data movements as crucial to be able to combine data from PE's that are located far apart from each other. For all these algorithm only the $\Theta(n)$-notation is given, factors are not presented. For most problems, the solutions do not extend to three dimensions.

Other architectures, like the pyramid or the mesh of trees, constructed from layers of mesh-connected computers, have been studied. For example, Schwarzkopf [Sch89] gave an algorithm to compute the discrete Voronoi diagram on the mesh of trees architecture. Miller and Stout [MS88] presented algorithms for identifying the extreme points of convex hulls of point sets for different architectures. These architectures differ from our pixel grid layout. To the best of our knowledge no principal component analysis algorithm has been presented.

Object Detection. We aim at detecting and describing the (potentially unknown) objects in a given B/W image. While we are interested in using distributed algorithms on a field of sensors, object detection, pattern recognition and object recognition (see e.g. Gonzalez and Woods [GW08]) are well-studied, important problems. All these are in the scope of computer vision (see Ballard and Brown [BB82]), which of course also involves precedent tasks as describing an image, and on this basis recognition, that is, recognizing local discontinuities in intensity for identifying edges and, more sophisticated, segments, shape and clusters. Object, pattern, and face recognition concentrates on certifying whether pre-defined or learned objects (patterns, faces) or object classes are present in the given image. Object identification deals with an even more specified task: within a given class (e.g., cars) a certain object is to be identified, see Ferencz et al. [FLMM04]. By contrast, we want to detect an unknown object (or objects) and determine, for example,

its center of gravity. While the task of distinguishing the object from the background is—as we face monochrome images—not our focus of interest.

Principal Component Analysis. For the detected objects we are interested in several attributes, like centroid position and orientation. We consider the pixels as a point set distribution in $\mathbb{R}^2$. Thus, these attributes can be described by moments. For given data sets, attributes like orientation can be determined with a Principal Component Analysis (PCA) (for an overview on PCA see Shlens [Shl05] or Joliffe [Jol02]). This analysis is widely used for statistical analysis of data, e.g., from neuroscience, from gene expression data (see Wall et al. [WRR03]), as well as in computer vision, for both representation (see Gonzalez and Woods [GW08]) and image compression.

Determining the Eigenvectors of the covariance matrix and ordering these by the related Eigenvalue (highest to lowest), gives the components in order of significance. That is, for visualized data with coordinates the first principal component gives the orientation.

Distributed Model. As described above, we assume the $\mathcal{LOCAL}$ model of Peleg [Pel00] for the communication between pixels. In contrast to the $\mathcal{LOCAL}$ model, the other two representative models defined by Peleg, the $\mathcal{CONGEST}$ and the $\mathcal{ASYNC}$ model, focus on volume limitations and asynchrony, respectively. Thus, instead of allowing for messages of arbitrary size, as does the $\mathcal{LOCAL}$ model, the $\mathcal{CONGEST}$ model limits the message size to $O(\log n)$. The $\mathcal{ASYNC}$ model assumes asynchronous communication.

Distributed Algorithms. We present a distributed sweep over the objects. If one is not aware of the special structure of the problem, a first approach would be to use a standard distributed algorithm on the graph induced by the object pixels. Thus, instead of using our algorithm we could use any leader election or spanning tree algorithm, see Elkin [Elk06], Gallager et al. [GHS83] and Awerbuch et al. [Awe87, ABCP96], as well as Lynch [Lyn96] for an overview. So, this would be a distributed algorithm operating on object pixels only (sending no messages via non-object pixels). Such a distributed in-object algorithm can not have a runtime better than $\Omega(WH)$ for objects on a $W \times H$ pixel image, because there exist objects with a diameter of $\Omega(WH)$, see Figure 3.3. Our algorithm utilizes the correlation between input geometry and network geometry, using non-object pixels as well as object pixels. This approach yields a runtime complexity of $O(W + H)$.

3.2 Computation of Attributes

As stated in the preceding section we intend to compute certain attributes of an object in the image $\mathcal{X}$. Let $\mathcal{X}' \subseteq \mathcal{X}$ be an object in $\mathcal{X}$. We want to

compute the following attributes:

1. The size $|\mathcal{X}'|$ of the object. (That is, the number of connected black pixels forming the object, using the connectivity definition given in Section 3.1.1.)

2. The center of gravity $(\mu_{\mathrm{x}}, \mu_{\mathrm{y}})$ of $\mathcal{X}'$. With (x_p, y_p) being the coordinates of pixel p in some global coordinate system (that is unknown to the pixels), we have:

$$\mu_{\mathrm{x}} = \frac{1}{|\mathcal{X}'|} \sum_{p \in \mathcal{X}'} x_p \ , \quad \mu_{\mathrm{y}} = \frac{1}{|\mathcal{X}'|} \sum_{p \in \mathcal{X}'} y_p \ . \tag{3.1}$$

3. The orientation of the object. We compute the second moments in the form of the covariance matrix

$$\begin{pmatrix} \sigma_{\mathrm{x}}^2 & \sigma_{\mathrm{xy}} \\ \sigma_{\mathrm{xy}} & \sigma_{\mathrm{y}}^2 \end{pmatrix}, \tag{3.2}$$

with

$$\sigma_{\mathrm{x}}^2 = \frac{1}{|X|} \sum_{p \in X} (x_p - \mu_{\mathrm{x}})^2, \tag{3.3}$$

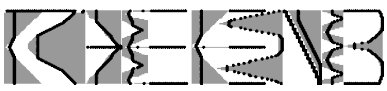
$$\sigma_{\mathrm{y}}^2 = \frac{1}{|X|} \sum_{p \in X} (y_p - \mu_{\mathrm{y}})^2, \tag{3.4}$$

$$\sigma_{\mathrm{xy}} = \frac{1}{|X|} \sum_{p \in X} (x_p - \mu_{\mathrm{x}})(y_p - \mu_{\mathrm{y}}), \tag{3.5}$$

considering the pixels as a point set in $\mathbb{R}^2$. For this two-dimensional point set two Principal Component Axes (orthogonal to each other) are defined: the Eigenvectors of this matrix; see Shlens [Shl05]. The Eigenvector corresponding to the largest Eigenvalue gives the direction of largest variance. We use this value as the orientation of the object.

Thus, for computing the desired attributes we need to determine the moments of the point set defined by the pixels.

If we would have some global processor with global knowledge of the point set, we could compute these values directly. But we only have pixels with local knowledge. Therefore we use a sweep of agents. We assign a "weight" of 1 to each object pixel. The agents, messages sent from a pixel to a neighboring pixel, "consume" these weights. The weight of a pixel can be consumed only once. During the sweep we cumulate relative weights. When



the algorithm finishes all cumulated relative weight is forwarded to a single pixel.

So, we need to define a message type send by the pixels (messages "carried" along by the agents), that give relative information and allows for the computation of the desired attributes in the end. In fact, an agent carries a six-tuple—an agent on a pixel (x, y) that consumed the weights for object pixels $X \subseteq \mathcal{X}'$ carries the tuple S:

$$\begin{aligned} S &= (m, s_x, s_y, s_{xx}, s_{yy}, s_{xy}) \\ &= (|X|, \sum_{p\in X}(x - x_p), \sum_{p\in X}(y - y_p), \sum_{p\in X}(x - x_p)^2, \\ &\qquad \sum_{p\in X}(y - y_p)^2, \sum_{p\in X}(x - x_p)(y - y_p)) \end{aligned} \tag{3.6}$$

These values are relative: they are centered in the pixel that send it (and will be adapted to be centered in the receiving pixel).

This six-tuple allows for the computation of the desired attributes and is easily updated when passed on to an adjacent pixel—in the following we show these and some other properties of the tuple.

Claim 1. *The values in the tuple S are sufficient for the computation of the aforementioned moments, if (x, y) is known. We assume that a centralized processor will pick up S from the distinguished pixel per object after the algorithm ended. Consequently, (x, y) can be accessed by this processor. The centralized processor then computes the moments.*

Proof. We show how the terms from Equations 3.1-3.5 can be expressed in terms of the entries of S:

$$\begin{aligned} \mu_{\mathrm{x}} &= \frac{1}{|X|}\sum_{p\in X} x_p = \frac{1}{|X|}\sum_{p\in X}(x - (x - x_p)) \\ &= \frac{1}{|X|}(-s_{\mathrm{x}} + |X|x) = x - \frac{s_{\mathrm{x}}}{|X|} = x - \frac{s_{\mathrm{x}}}{m}\,, \end{aligned} \tag{3.7}$$

$$\begin{aligned} \mu_{\mathrm{y}} &= \frac{1}{|X|}\sum_{p\in X} y_p = \frac{1}{|X|}\sum_{p\in X}(y - (y - y_p)) \\ &= \frac{1}{|X|}(-s_{\mathrm{y}} + |X|y) = y - \frac{s_{\mathrm{y}}}{|X|} = y - \frac{s_{\mathrm{y}}}{m}\,, \end{aligned} \tag{3.8}$$

$$
\begin{aligned}
\sigma_{\mathrm{x}}^2 &= \frac{1}{|X|}\sum_{p\in X}(x_p-\mu_{\mathrm{x}})^2 = \frac{1}{|X|}\sum_{p\in X}(x_p - x + \frac{1}{|X|}s_{\mathrm{x}})^2 \\
&= \frac{1}{|X|}\sum_{p\in X}((x_p-x)^2 + 2\frac{1}{|X|}(x_p-x)s_{\mathrm{x}} + \frac{s_{\mathrm{x}}^2}{|X|^2}) \\
&= \frac{1}{|X|}(s_{\mathrm{xx}} + 2\frac{1}{|X|}s_{\mathrm{x}}\sum_{p\in X}(x_p-x) + |X|\frac{s_{\mathrm{x}}^2}{|X|^2}) \\
&= \frac{1}{|X|}(s_{\mathrm{xx}} - \frac{2}{|X|}s_{\mathrm{x}}^2 + \frac{s_{\mathrm{x}}^2}{|X|}) = \frac{s_{\mathrm{xx}}}{|X|} - \frac{s_{\mathrm{x}}^2}{|X|^2} = \frac{s_{\mathrm{xx}}}{m} - \frac{s_{\mathrm{x}}^2}{m^2}\,, \qquad (3.9) \\
\sigma_{\mathrm{y}}^2 &= \frac{1}{|X|}\sum_{p\in X}(y_p-\mu_{\mathrm{y}})^2 = \frac{1}{|X|}\sum_{p\in X}(y_p - y + \frac{1}{|X|}s_{\mathrm{y}})^2 \\
&= \frac{1}{|X|}\sum_{p\in X}((y_p-y)^2 + 2\frac{1}{|X|}(y_p-y)s_{\mathrm{y}} + \frac{s_{\mathrm{y}}^2}{|X|^2}) \\
&= \frac{1}{|X|}(s_{\mathrm{yy}} + 2\frac{1}{|X|}s_{\mathrm{y}}\sum_{p\in X}(y_p-y) + |X|\frac{s_{\mathrm{y}}^2}{|X|^2}) \\
&= \frac{1}{|X|}(s_{\mathrm{yy}} - \frac{2}{|X|}s_{\mathrm{y}}^2 + \frac{s_{\mathrm{y}}^2}{|X|}) = \frac{s_{\mathrm{yy}}}{|X|} - \frac{s_{\mathrm{y}}^2}{|X|^2} = \frac{s_{\mathrm{yy}}}{m} - \frac{s_{\mathrm{y}}^2}{m^2}\,, \qquad (3.10) \\
\sigma_{\mathrm{xy}} &= \frac{1}{|X|}\sum_{p\in X}(x_p-\mu_{\mathrm{x}})(y_p-\mu_{\mathrm{y}}) \\
&= \frac{1}{|X|}\sum_{p\in X}(x_p - x + \frac{1}{|X|}s_{\mathrm{x}})(y_p - y + \frac{1}{|X|}s_{\mathrm{y}}) \\
&= \frac{1}{|X|}(\sum_{p\in X}(x_p-x)(y_p-y) + \frac{1}{|X|}s_{\mathrm{y}}\sum_{p\in X}(x_p-x) \\
&\quad + \frac{1}{|X|}s_{\mathrm{x}}\sum_{p\in X}(y_p-y) + |X|\frac{s_{\mathrm{x}}s_{\mathrm{y}}}{|X|^2}) \\
&= \frac{1}{|X|}(s_{\mathrm{xy}} - \frac{1}{|X|}s_{\mathrm{y}}s_{\mathrm{x}} - \frac{1}{|X|}s_{\mathrm{x}}s_{\mathrm{y}} + \frac{1}{|X|}s_{\mathrm{x}}s_{\mathrm{y}}) \\
&= \frac{s_{\mathrm{xy}}}{|X|} - \frac{s_{\mathrm{x}}s_{\mathrm{y}}}{|X|^2} = \frac{s_{\mathrm{xy}}}{m} - \frac{s_{\mathrm{x}}s_{\mathrm{y}}}{m^2}\,, \qquad (3.11)
\end{aligned}
$$

So, we can compute all entries of the covariance matrix (3.2). The Eigenvector for the larger Eigenvalue, the direction of maximum variance, can then be easily computed by the global observer processor.

□

We showed that S allows for the computation of the desired attributes. Moreover, we are only allowed to perform simple computations with a pixel. We can prove that addition and subtraction suffice for updates after an agent's move:

Claim 2. *When an agent moves from (x, y) to an adjacent pixel, addition and subtraction are sufficient for all necessary updates of the tuple for the new position.*

(i) When moving a tuple $(m, s_x, s_y, s_{xx}, s_{yy}, s_{xy})$ to the east, $(m, s_x+m, s_y, s_{xx}+2s_x+m, s_{yy}, s_{xy}+s_y)$ is the correct tuple for the new position.

(ii) When moving a tuple $(m, s_x, s_y, s_{xx}, s_{yy}, s_{xy})$ to the west, $(m, s_x-m, s_y, s_{xx}-2s_x+m, s_{yy}, s_{xy}-s_y)$ is the correct tuple for the new position.

(iii) When moving a tuple $(m, s_x, s_y, s_{xx}, s_{yy}, s_{xy})$ to the north, $(m, s_x, s_y+m, s_{xx}, s_{yy}+2s_y+m, s_{xy}+s_x)$ is the correct tuple for the new position.

(iv) When moving a tuple $(m, s_x, s_y, s_{xx}, s_{yy}, s_{xy})$ to the south, $(m, s_x, s_y-m, s_{xx}, s_{yy}-2s_y+m, s_{xy}-s_x)$ is the correct tuple for the new position.

Proof. We consider the different updates

(i) Assume that pixel (x, y) gets a message $S = (m, s_x, s_y, s_{xx}, s_{yy}, s_{xy})$ from the left, i.e., from $(x-1, y)$. The receiver has to compute the following values to ensure the properties of the tuple:

$$\begin{aligned}
s'_{\mathrm{x}} &= \sum_{p\in X}(x-x_p) = \sum_{p\in X}(x-1-x_p) + \sum_{p\in X}1 \\
&= s_{\mathrm{x}} + m \ , \\
s'_{\mathrm{y}} &= \sum_{p\in X}(y-y_p) = s_{\mathrm{y}} \ , \\
s'_{\mathrm{xx}} &= \sum_{p\in X}(x-x_p)^2 = \sum_{p\in X}((x-1-x_p)+1)^2 \\
&= \sum_{p\in X}(x-1-x_p)^2 \ + \ 2\sum_{p\in X}(x-1-x_p) \ + \ \sum_{p\in X}1 \\
&= s_{\mathrm{xx}} + 2s_{\mathrm{x}} + m \ , \\
s'_{\mathrm{yy}} &= \sum_{p\in X}(y-y_p)^2 = s_{\mathrm{yy}} \ , \\
s'_{\mathrm{xy}} &= \sum_{p\in X}(x-x_p)(y-y_p) = \sum_{p\in X}(x-1-x_p+1)(y-y_p) \\
&= \sum_{p\in X}(x-1-x_p)(y-y_p) + \sum_{p\in X}(y-y_p) \\
&= s_{\mathrm{xy}} + s_{\mathrm{y}} \ .
\end{aligned}$$

(ii) Assume that pixel (x, y) gets a message $S = (m, s_x, s_y, s_{xx}, s_{yy}, s_{xy})$ from the right, i.e., from $(x+1, y)$. The feasible values for the receiver are:

$$\begin{aligned}
s'_{\mathrm{x}} &= \sum_{p\in X}(x - x_p) = \sum_{p\in X}(x + 1 - x_p) - \sum_{p\in X} 1 \\
&= s_{\mathrm{x}} - m \ , \\
s'_{\mathrm{y}} &= \sum_{p\in X}(y - y_p) = s_{\mathrm{y}} \ , \\
s'_{\mathrm{xx}} &= \sum_{p\in X}(x - x_p)^2 = \sum_{p\in X}((x + 1 - x_p) - 1)^2 \\
&= \sum_{p\in X}(x + 1 - x_p)^2 \ - \ 2\sum_{p\in X}(x + 1 - x_p) \ + \ \sum_{p\in X} 1 \\
&= s_{\mathrm{xx}} - 2s_{\mathrm{x}} + m \ , \\
s'_{\mathrm{yy}} &= \sum_{p\in X}(y - y_p)^2 = s_{\mathrm{yy}} \ , \\
s'_{\mathrm{xy}} &= \sum_{p\in X}(x - x_p)(y - y_p) = \sum_{p\in X}(x + 1 - x_p - 1)(y - y_p) \\
&= \sum_{p\in X}(x + 1 - x_p)(y - y_p) - \sum_{p\in X}(y - y_p) \\
&= s_{\mathrm{xy}} - s_{\mathrm{y}} \ .
\end{aligned}$$

(iii) Assume that pixel (x, y) gets a message $S = (m, s_x, s_y, s_{xx}, s_{yy}, s_{xy})$ from below, i.e., from $(x, y-1)$. The receiver computes the following values:

$$\begin{aligned}
s'_{\mathrm{x}} &= \sum_{p\in X}(x - x_p) = s_{\mathrm{x}} \ , \\
s'_{\mathrm{y}} &= \sum_{p\in X}(y - y_p) = \sum_{p\in X}(y - 1 - y_p) + \sum_{p\in X} 1 \\
&= s_{\mathrm{y}} + m \ , \\
s'_{\mathrm{xx}} &= \sum_{p\in X}(x - x_p)^2 = s_{\mathrm{xx}} \ , \\
s'_{\mathrm{yy}} &= \sum_{p\in X}(y - y_p)^2 = \sum_{p\in X}((y - 1 - y_p) + 1)^2 \\
&= \sum_{p\in X}(y - 1 - y_p)^2 \ + \ 2\sum_{p\in X}(y - 1 - y_p) \ + \ \sum_{p\in X} 1 \\
&= s_{\mathrm{yy}} + 2s_{\mathrm{y}} + m \ ,
\end{aligned}$$

$$\begin{aligned} s'_{\mathrm{xy}} &= \sum_{p\in X}(x-x_p)(y-y_p) = \sum_{p\in X}(x-x_p)(y-1-y_p+1) \\ &= \sum_{p\in X}(x-x_p)(y-1-y_p) + \sum_{p\in X}(x-x_p) \\ &= s_{\mathrm{xy}} + s_{\mathrm{x}} \ . \end{aligned}$$

(iv) Assume that pixel (x, y) gets a message $S = (m, s_x, s_y, s_{xx}, s_{yy}, s_{xy})$ from above, i.e., from $(x, y+1)$. The receiver computes the following values:

$$\begin{aligned} s'_{\mathrm{x}} &= \sum_{p\in X}(x-x_p) = s_{\mathrm{x}} \ , \\ s'_{\mathrm{y}} &= \sum_{p\in X}(y-y_p) = \sum_{p\in X}(y+1-y_p) - \sum_{p\in X}1 \\ &= s_{\mathrm{y}} - m \ , \\ s'_{\mathrm{xx}} &= \sum_{p\in X}(x-x_p)^2 = s_{\mathrm{xx}} \ , \\ s'_{\mathrm{yy}} &= \sum_{p\in X}(y-y_p)^2 = \sum_{p\in X}((y+1-y_p)-1)^2 \\ &= \sum_{p\in X}(y+1-y_p)^2 \ - \ 2\sum_{p\in X}(y+1-y_p) \ + \ \sum_{p\in X}1 \\ &= s_{\mathrm{yy}} - 2s_{\mathrm{y}} + m \ , \\ s'_{\mathrm{xy}} &= \sum_{p\in X}(x-x_p)(y-y_p) = \sum_{p\in X}(x-x_p)(y+1-y_p-1) \\ &= \sum_{p\in X}(x-x_p)(y+1-y_p) - \sum_{p\in X}(x-x_p) \\ &= s_{\mathrm{xy}} - s_{\mathrm{x}} \ . \end{aligned}$$

Only basic arithmetic operations, that is, addition and subtraction, are used for the updates. □

Claim 3. *For a pixel field of size $2^k \times 2^k$ we need at most $20k + 11$ bits for a tuple.*

Proof. Let us consider a pixel field of size $2^k \times 2^k$. Obviously, we have $m \leq 2^k \times 2^k$. Consequently, we need at most $2k + 1$ bits for m. Furthermore, $|s_{\mathrm{x}}| \leq 2^k \sum_{i=1}^{2^k} i \leq 2^{3k}$ (analogue for s_{y}), thus, we need at most $3k + 2$ for the second and third entry in the tuple (in each case one bit for the sign and $3k + 1$ for the value). For the last three entries we can give an

upper bound of $2^k \sum_{i=1}^{2^k} i^2 \leq 2^{4k}$, resulting in an upper bound of $4k+2$ on the number of bits used for these entries. Altogether, we need at most $2k+1+2(3k+2)+3(4k+2) = 20k+11$ bits for a tuple. □

So far we considered the necessary updates when an agent moves. We still need to make sure that after the necessary updates described in the proof of Claim 2, weight from the new pixel is consumed, if it is still available.

Claim 4. *If an agent picks up the weight of the pixel it currently resides on, it simply has to increment the first tuple entry.*

Proof. For a pixel p whose weight has not yet been consumed, we have $p \in X$. After the updates of the other tuple entries $m = |X \backslash p| = |X| - 1$. Hence, incrementing the first tuple entry yields $|X|$. □

Claim 5. *In case six-tuples arrive from different directions, we simply add them component-wise. If an agent receives the cumulative weight of another agent on the same pixel, it just computes the component-wise sum of the two tuples.*

Proof. Assume, e.g., that pixel (x, y) gets a message from below, i.e., from $(x, y-1)$, and a message from above, i.e., from $(x, y+1)$. The message from $(x, y-1)$ and from $(x, y+1)$ refers to an agent that consumed the weights for object pixels $X_1 \subseteq \mathcal{X}'$ and $X_2 \subseteq \mathcal{X}'$, respectively. Hence, we may simply split the sums over all $p \in X$ into sums over all $p \in X_1$ and over all $p \in X_2$. □

In order to clarify these computations we have a look at the example of Figure 3.6. The left side depicts the pixel image, on the right side the cumulation of relative weights is shown. After the sweep has stopped, the centralized processor collects the tuple $(6, 7, -5, 11, 9, -7)$ from the pixel at $(32, 2)$. Using (3.7) to (3.11), the processor computes the following values:

$$\begin{aligned}
\mu_{\mathrm{x}} &= x - \frac{s_{\mathrm{x}}}{m} = 30 + \frac{5}{6}\,, \\
\mu_{\mathrm{y}} &= y - \frac{s_{\mathrm{y}}}{m} = 2 + \frac{5}{6}\,, \\
\sigma_{\mathrm{x}}^2 &= \frac{s_{\mathrm{xx}}}{m} - \frac{s_{\mathrm{x}}^2}{m^2} = \frac{17}{36}\,, \\
\sigma_{\mathrm{y}}^2 &= \frac{s_{\mathrm{yy}}}{m} - \frac{s_{\mathrm{y}}^2}{m^2} = \frac{29}{36}\,, \\
\sigma_{\mathrm{xy}} &= \frac{s_{\mathrm{xy}}}{m} - \frac{s_{\mathrm{x}} s_{\mathrm{y}}}{m^2} = \frac{-7}{36}\,.
\end{aligned}$$

Thus, the center is located at $(30+\frac{5}{6}, 2+\frac{5}{6})$. The resulting covariance matrix is

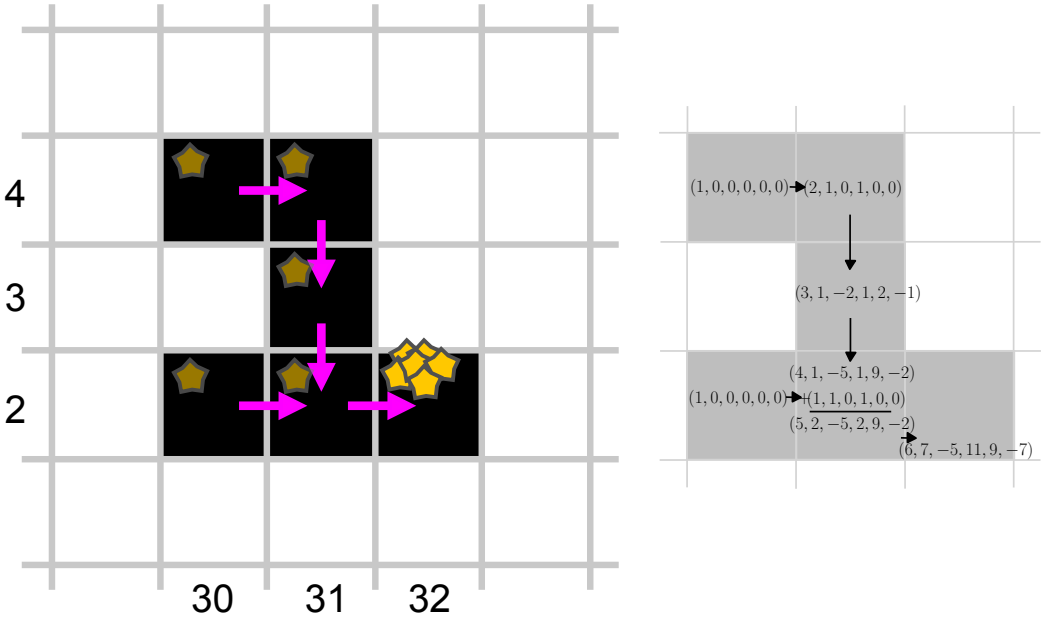


Figure 3.6: Example for the scheme to accumulate relative weights. The yellow stars indicate the consumed weights of the pixels.

$$\begin{pmatrix} \frac{17}{36} & \frac{-7}{36} \\ \frac{-7}{36} & \frac{29}{36} \end{pmatrix}.$$

The Eigenvector for the largest Eigenvalue is

$$(\frac{29}{7} + \frac{1}{7}(-23 - \sqrt{85}), 1) \approx (-0.459935, 1),$$

giving the direction of maximum variance, the orientation.

Now, the only task we are left with is to define a rule set for the sweep. This rule set must describe how the agents sweep over the objects, how they consume the pixel weights and then let the weights for each object accumulate in a single pixel. We will describe this rule set in the next section.

3.3 Our Algorithm

We give a rule set describing an agents sweep over the objects. In order to do so, we will first give some definitions for these agents in Section 3.3.1. We give the general idea for the sweep and the rule set in the following Section 3.3.2 and end the algorithm's description with a note on its implementation.

3.3.1 Agents

We let A denote the set of all agents. Aside from the six variables for computing the moments, described in Section 3.2, an agent $a \in A$ has two more variables:

1. $C_a \subseteq A$. This denotes the set of partner agents.
2. h_a. This boolean variable indicates whether the agent a has ever moved onto a non-object pixel.

Partner agents belong to the same object as a, that is, they started in the same object, and reside in an adjacent row (within L_∞-distance of 1). Once established, partnerships will never be broken. To assure this, partner agents move together, maintaining an L_∞-distance of at most 1, waiting for each other if necessary. Initially, an agent has no partners, so, we start with $C_a = \varnothing$ for all agents. Whenever an agent a decides to add a' to its set of partners, a' will also decide to add a; that is, partnership is mutual. Consequently, we may consider an established partnership between two agents to be an undirected edge connecting the two agents. With the agents as vertices we may, thus, consider partnership as an undirected graph on the set of agents. We denote this graph by $\mathcal{C} = (A, C)$. Using these partnerships allows us to deal with interwoven objects: Information is exchanged with partners only, so even if agents originating from different objects occupy the same pixel, we will never mix the information, since such agents never become partners.

The boolean variable h_a indicates whether the agent a has ever moved on a non-object pixel and, hence, left the object it started in, its "home" object. Initially, we set h_a to false for all agents (which are only created on object pixels). Once the agent a leaves its home object and moves to a non-object pixel, it sets h_a to true.

Having defined the variables used by an agent, we consider the information available to an agent. Let $A(p) \subseteq A$ denote the set of all agents currently located on a pixel p. We assume that each agent a residing on a pixel p knows

- Which pixels $q \in \overline{\Gamma}(p)$ are object pixels
- If the weight of p has already been consumed
- The variables $C_{a'}$ and $h_{a'}$ for all $a' \in \cup_{q \in \overline{\Gamma}(p)} A(q)$.

We show how an agent may actually access this information on a smart pixel grid in Section 3.3.2.

3.3.2 Algorithm

Basic Idea. Before we give the formal rule set, we have a look at its basic ideas. We start with agents at the left boundary of objects (that is, initially there is an agent on every pixel that has a non-object pixel to its left). The agents establish partnerships with agents from the same home object. These chains of partner agents sweep over the object, from the left object boundary to the point when they reach a vertical line at the object's rightmost coordinate. Once established, partnerships will never be broken again. During this sweep the weight assigned to object pixels is collected by agents that never left their home object. These collected weights are treated as the six-tuple S described in Section 3.2. Since we intend to have a single pixel with all information on the moment computation, we forward the cumulative weight downward, such that in the final vertical line the bottommost agent carries the desired information. How do we assure the realization of this general idea?

Initialization. Agents are initialized on every object pixel that has a non-object pixel to its left; see Figure 3.7. Those agents did not yet leave their home object and did not establish any partnership (i.e., we start with $h_a = \text{false}$ and $C_a = \varnothing$ for all agents a). The algorithm then runs in iterations, with all agents performing the same steps in each iteration.

Partners. The sweep proceeds from left to right, so in each iteration an agent looks for potential partners in vertical direction, that is, in adjacent rows. If there is an agent on a pixel in the accessible neighborhood $\Gamma(p)$ and in the same column as p with a y-coordinate different from p's, that never left its home object, a partnership is established. Selection of agents that are still in their home object assures the ability to cope with interwoven objects, as described in Section 3.3.1.

Weight Consumption and Forwarding. In each iteration agents that did not leave their home object consume a pixel's weight, in case it is still available. The downward forwarding of cumulative weight translates to an agent forwarding all collected cumulative weight that it currently carries to a partner agent in the row below, if such an agent exists (in each iteration, again).

Merging of Agents. Starting with agents at the left object boundary, we may have different agents belonging to the same object in a single row—think of a $\cap$-like object. Those will not become partners—one of the agents must have left its home object when it "meets" the other, and partner agents are searched for in adjacent rows (not the same row) only. Still, we want to merge agents in one row that belong to the same object. We use their partner agents here: If two agents a and a' on a pixel p have a common partner a^*,

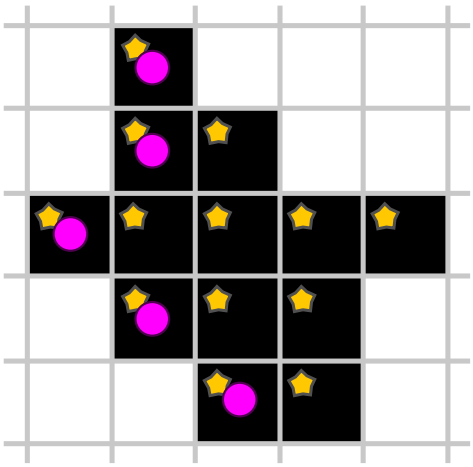


Figure 3.7: Example for an object: agents are located on every object pixel that has no object pixel to its left. The pink color indicates $h_a = \text{false}$. Every object pixel has a weight of 1, depicted by the yellow stars.

a and a^* as well as a' and a^* belong to the same object. Hence, a and a' may conclude that they have the same home object and decide to merge into a new agent a''. A merge includes an update of $h_{a''}$ and $C_{a''}$. Only if both agents already left their home object the new agent a'' is no longer allowed to collect weight and establish partnerships. In case either a or a' never left their object, the new agents inherits this status: $h_{a''} = h_a \wedge h_{a'}$. Moreover, we simply merge the set of partners.

Movement. The description above did not contain any movement, we intend to give a rule set for a sweep, hence, we need to clarify when an agent is allowed to move to the right (that is from p to $\mathsf{E}(p)$). We do so in naming all cases that prevent an agent from being allowed to move. We demanded that, once established, a partnership should never be broken again. Thus, if an agent a (on p) has a partner agent a' in the pixel to the north-west ($\mathsf{NW}(p)$) or to the south-west ($\mathsf{SW}(p)$) it is not allowed to move. In case it would transfer itself to $\mathsf{E}(p)$ and a' does not move in this iteration, the L_∞-distance between the two agents would become larger than 1. Moreover, agents with an object pixel to the south ($\mathsf{S}(p)$) that did not yet find a partner on $\mathsf{S}(p)$ do not move. They "wait" for an agent to arrive at this point of connection. The same happens for an object pixel to the north. The final condition that prevents an agent from moving applies to agents that have a non-object pixel to the right. In case there is no partner agent on either $\mathsf{NE}(p)$ or $\mathsf{SE}(p)$, which would "drag" the agent further to the right, the agent is not allowed to move to this non-object pixel. We will prove in Section 3.4.1 that this assures the vertical line at the largest x-coordinate of the object.

We now give the technical description of what we described above. Initially, an agent a is located on every object pixel that has no object pixel to its left. For each agent we have $h_a = \text{false}$ and $C_a = \varnothing$ (as it has not left its home object and not established any partnerships), see Figure 3.7.

The algorithm runs in iterations. The algorithm's steps are performed in sync by all agents in each iteration. We give the steps for agent a on pixel p (Figure 3.8 shows examples for the steps) in Algorithm 3.1.

A stopping criterion is not integrated in our algorithm. Yet, the algorithm stops. The algorithm ends in the first iteration in which no agent either

- Passes on cumulative weight in step 3
- Decides to move in step 5

The agents itself are not aware that the algorithm ended, as they only know whether they self performed any of these steps. For the application we may simply assume that a centralized processor waits for $2W + 2H$—upon which, as we will prove in Section 3.4.2, the algorithm terminated.

Examples for this algorithm are shown in Figures 3.9 to 3.13, Figure 3.14 and Figure 3.15. In Figures 3.9 to 3.13 the focus is on the details, while Figure 3.14 and Figure 3.15 lay the stress on the overall picture. Here, the collection of weights is not shown. In iteration #1, the agents are located on left-most object pixels. In the end the agents for each object build a vertical line. All cumulated weight for this object is carried by the bottommost agent and can be picked up. New partnerships are established, for example, in iteration #3 and #6 (Step 2). A merge can be observed in iteration #9: The two agents in the row second to the bottom have a common partner, in the bottommost row, and decide to merge when they are located on the same pixel. The new agent is still in its home object, the collected weights are simply merged. The agents only merge when they know that they belong to the same object. Thus, the chains for the different objects can cross each other without any interference in iterations #6 to #14. In addition, the different waiting rules can be observed. The uppermost agent of the vertical line of the "T" waits due to Rule 5d: there is a non-object pixel to the east and no agent in the north-east or south-east, this waiting condition is resolved in iteration #5. Rule 5a applies, e.g., for the bottommost agent in iteration #2. In the end the collected weight for the "T" and the "U" can be picked up from the bottommost pixels of the respective agent chains, as shown in Figure 3.13.

Figure 3.14 and Figure 3.15 show that agents move fast while they are on object pixels. After leaving an object, they have to be "dragged" according to Rule 5d by partner agents, see for example iterations #14 and #15.

Algorithm 3.1: Smart Pixels Sweep

1. Consume weight:
If $h_a = \text{false}$ and the weight on the pixel p has not been consumed yet, consume it (as described in Section 3.2), see Figure 3.8(1.).

2. Find partners:
If $h_a = \text{false}$, find out if there are any agents $a' \in \bigcup_{p' \in \Gamma(p): y_{p'} \neq y_p \& x_p = x_{p'}} A(p')$ (i.e., on neighbor pixels in adjacent rows), which also have $h_{a'} = \text{false}$ (i.e., which also never left their home object). Add those to C_a, i.e., establish a partnership with a'. See Figure 3.8(2.).

3. Pass on cumulative weights:
If there is an agent $a' \in C_{a'}$ in the row below p, pass all collected moment weight to a'. Ties are broken arbitrarily. See Figure 3.8(3.).

4. Merge agents:
If there is another agent $a' \in A(p)$ (i.e., on the same pixel as a), and a and a' have a common partner, they conclude they belong to the same object (as partnership is mutual). Then, they merge into a new agent a'' with $h_{a''} = h_a \wedge h_{a'}$ and $C_{a''} = C_a \cup C_{a'}$. This is done as a shrinking of $\{a, a'\}$ in $\mathcal{C}$, with the obvious consequences for the partner edges incident to them. The agents also merge the accumulated weights (see Section 3.2). See Figure 3.8(4.).

5. Decide whether to stay or go:
The agent decides to move forward, unless any of these conditions holds:

(a) $C_a \cap A(\mathsf{NW}(p)) \neq \varnothing$ or $C_a \cap A(\mathsf{SW}(p)) \neq \varnothing$ (i.e., it has a partner in the column to the left and could lose its partner by moving in case the partner will not move), see Figure 3.8(5.a), for the case of $C_a \cap A(\mathsf{NW}(p)) \neq \varnothing$,

(b) $\mathsf{S}(p) \in \mathcal{X}$, but $C_a \cap A(\mathsf{S}(p) \cup \mathsf{SW}(p) \cup \mathsf{SE}(p)) = \varnothing$ (i.e., there is an object pixel in the south, but no partner has been found on it yet), see Figure 3.8(5.b), or

(c) $\mathsf{N}(p) \in \mathcal{X}$, but $C_a \cap A(\mathsf{N}(p) \cup \mathsf{NW}(p) \cup \mathsf{NE}(p)) = \varnothing$ (analog to (b), here for the north).

(d) $\mathsf{E}(p) \notin \mathcal{X}$ and $C_a \cap A(\mathsf{NE}(p)) = C_a \cap A(\mathsf{SE}(p)) = \varnothing$ (i.e., it will not move onto the non-object pixel to the east until it gets dragged there by partner agents), see Figure 3.8(5.d).

6. Stay or go:
If a decided to move in step 5, it now transfers itself to $\mathsf{E}(p)$. If $\mathsf{E}(p) \notin \mathcal{X}$, it sets $h_a \leftarrow \text{true}$ (as it is no longer in its home object).

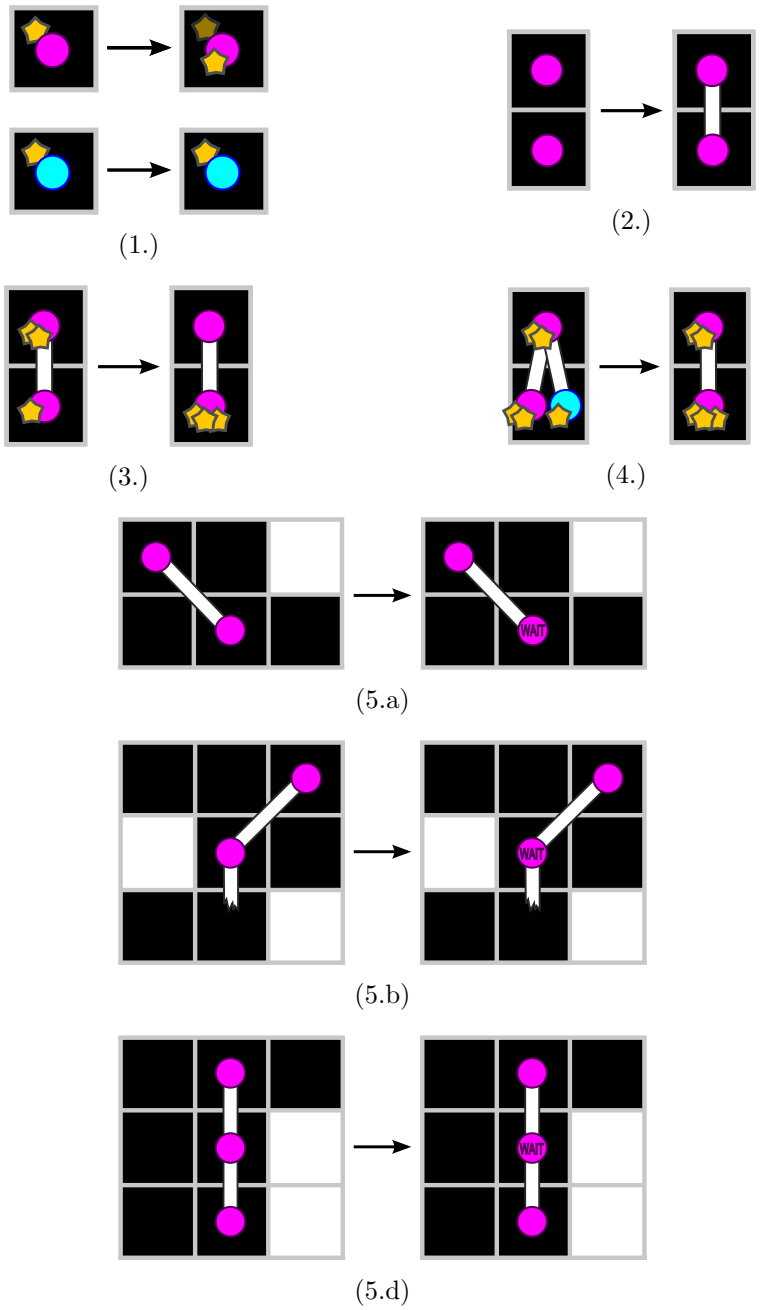


Figure 3.8: Examples for the steps in the algorithm. Agents that never left their home object are shown in pink, agents with h_a = true in turquoise. Yellow stars indicate pixel weight, they are drawn pale if the pixel's weight has already been consumed. The numbers refer to the steps in the algorithm.

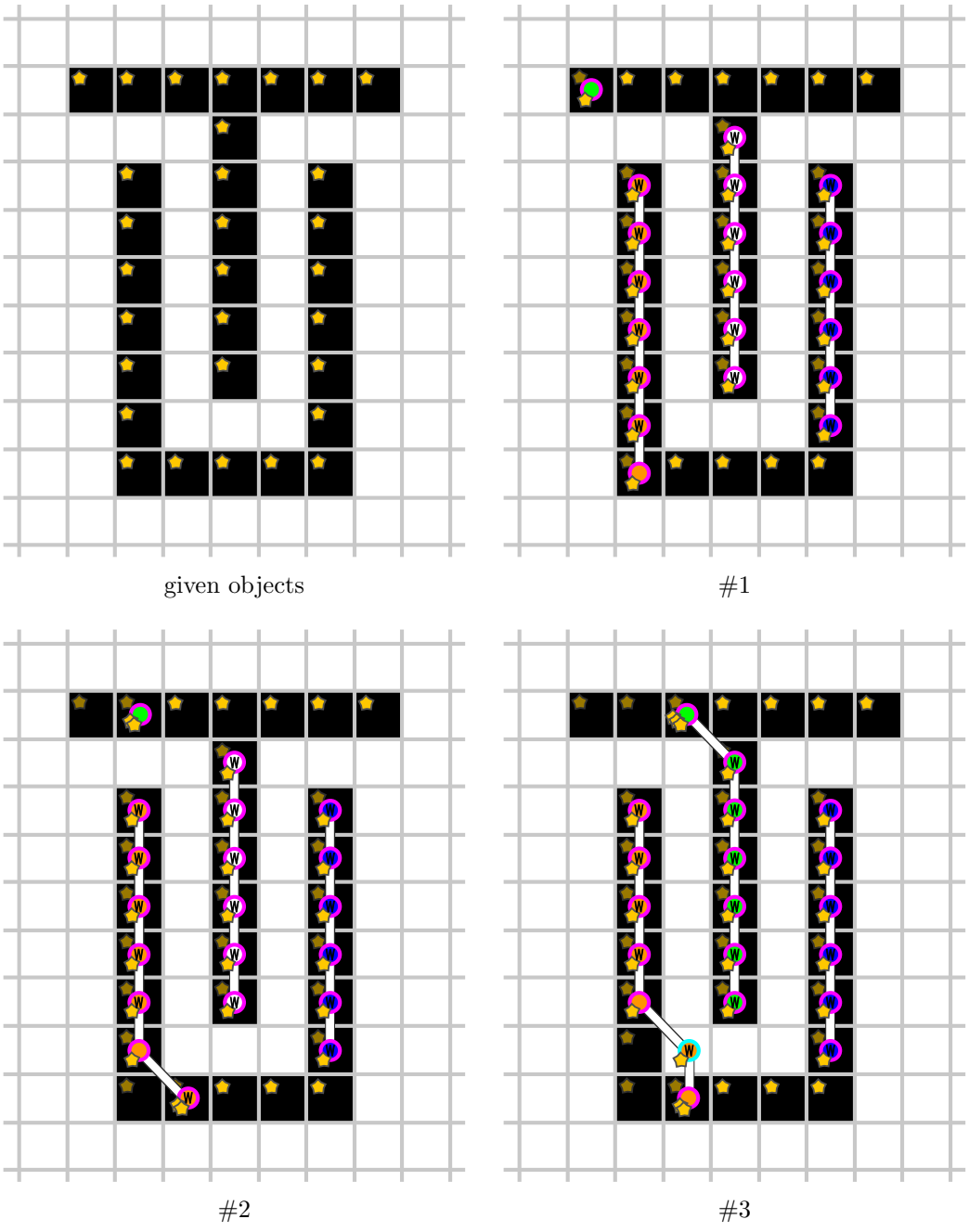


Figure 3.9: Example for algorithm 3.1. The pink color of the agents boundary indicates h_a = false, h_a = true is indicated in turquoise. Yellow stars indicate pixel weight; they are drawn pale if the pixel's weight has already been consumed. For clarity reasons the consumed weight is not forwarded during the run of the algorithm; these steps are united in the last picture. An agent labeled with a W has to wait in the current round.

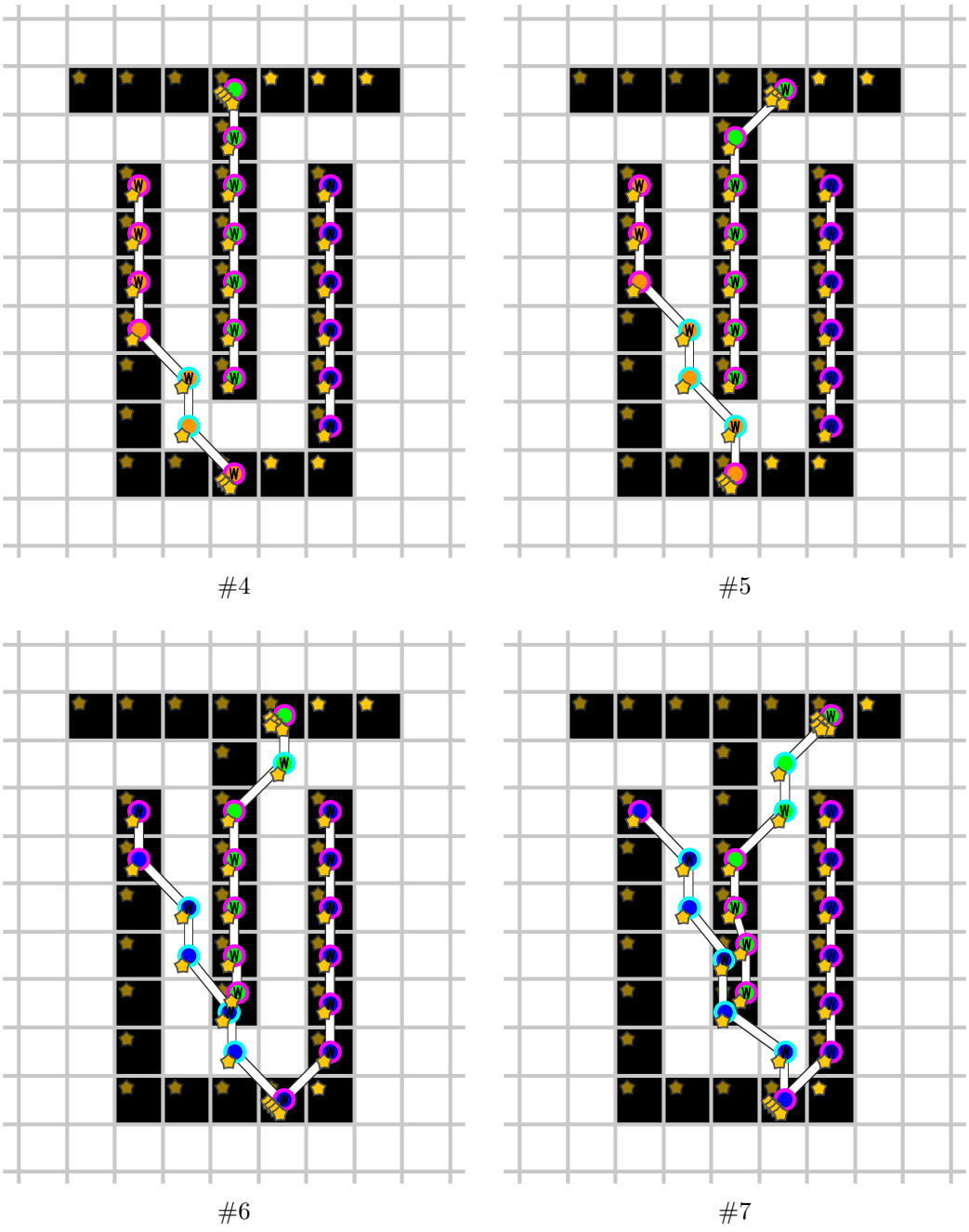


Figure 3.10: Example of Figure 3.9 continued. The pink color of the agents boundary indicates $h_a = \text{false}$, $h_a = \text{true}$ is indicated in turquoise. Yellow stars indicate pixel weight; they are drawn pale if the pixel's weight has already been consumed. For clarity reasons the consumed weight is not forwarded during the run of the algorithm; these steps are united in the last picture. An agent labeled with a W has to wait in the current round.

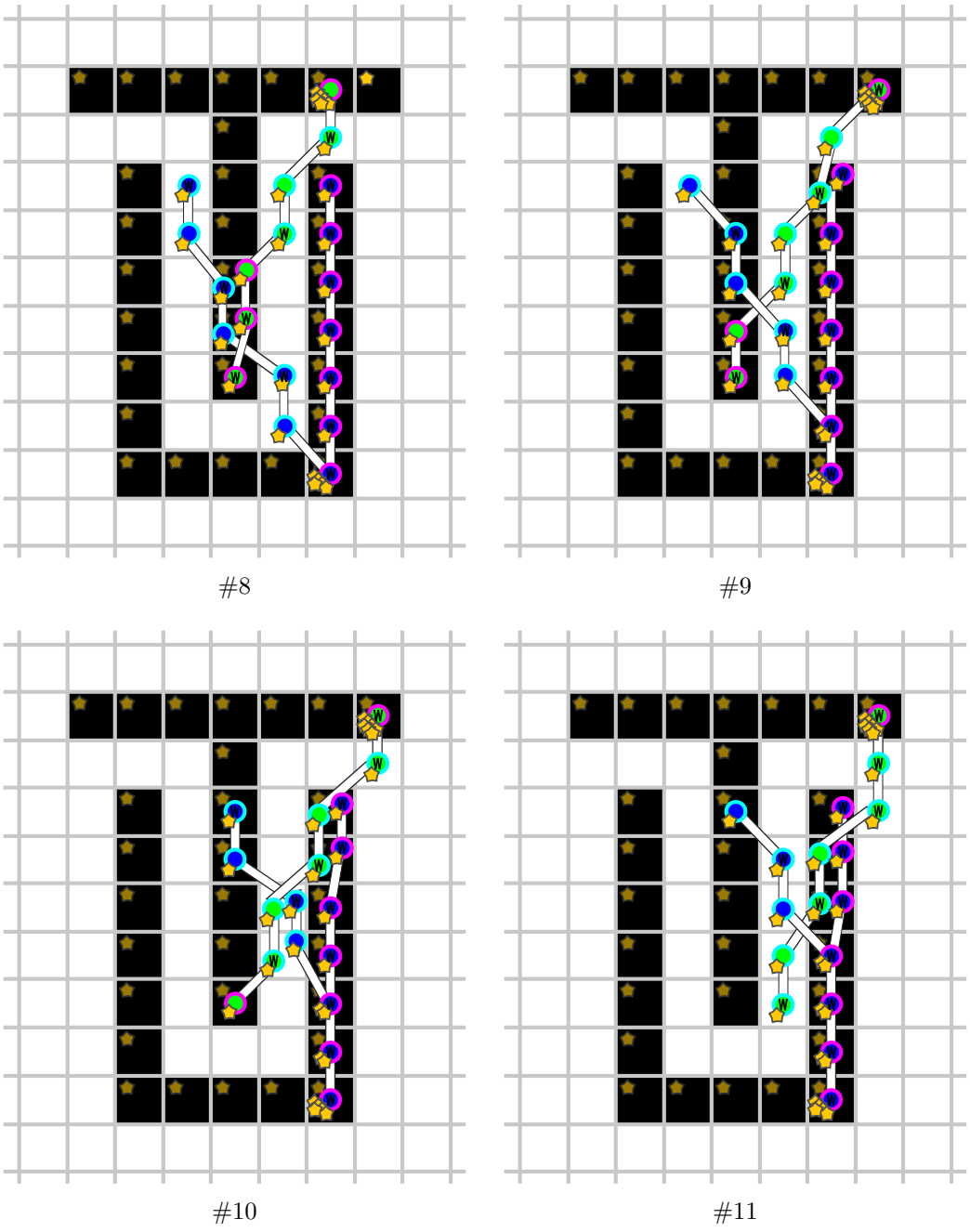


Figure 3.11: Example of Figure 3.9 continued. The pink color of the agents boundary indicates h_a = false, h_a = true is indicated in turquoise. Yellow stars indicate pixel weight; they are drawn pale if the pixel's weight has already been consumed. For clarity reasons the consumed weight is not forwarded during the run of the algorithm; these steps are united in the last picture. An agent labeled with a W has to wait in the current round.

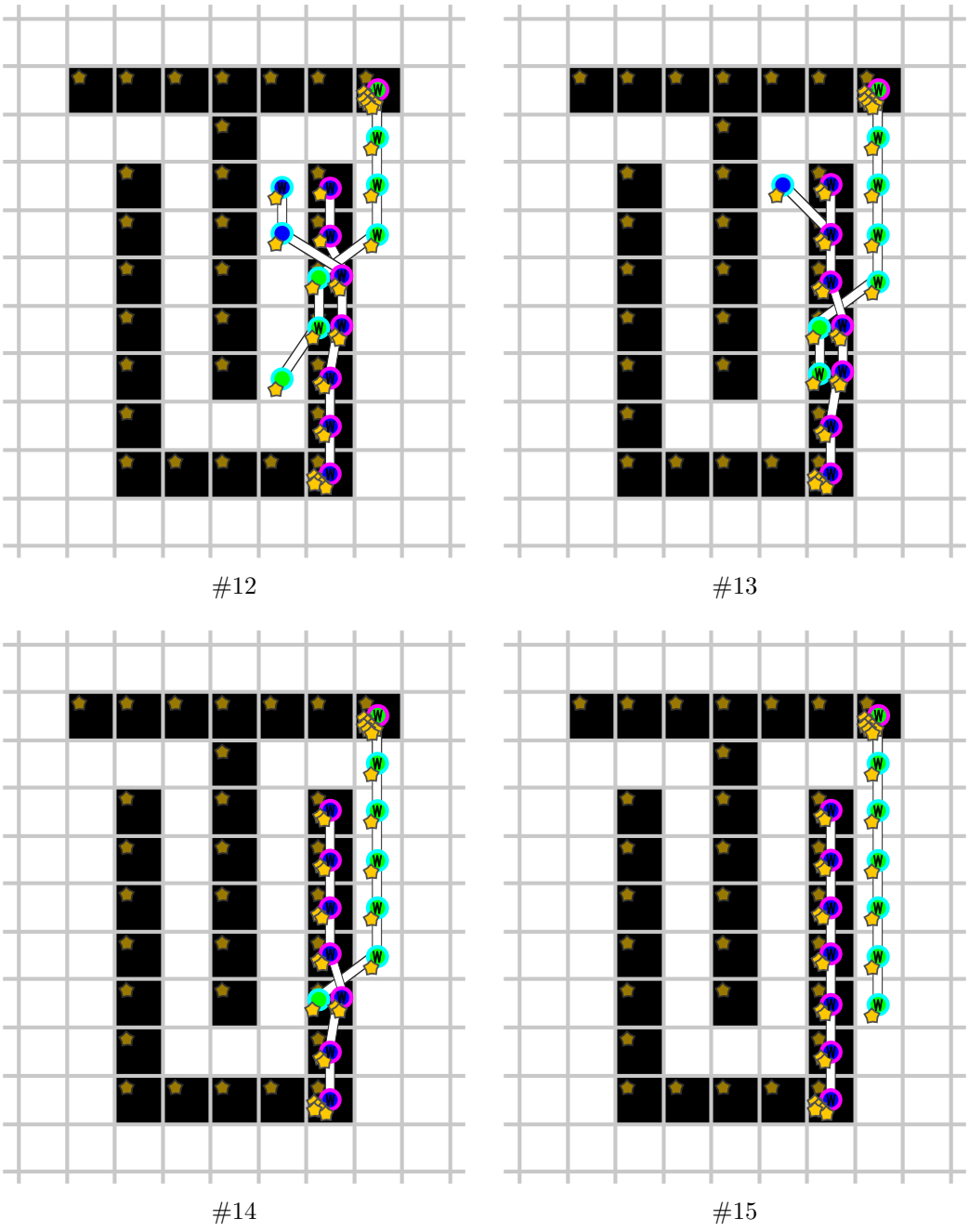


Figure 3.12: Example of Figure 3.9 continued. The pink color of the agents boundary indicates h_a = false, h_a = true is indicated in turquoise. Yellow stars indicate pixel weight; they are drawn pale if the pixel's weight has already been consumed. For clarity reasons the consumed weight is not forwarded during the run of the algorithm; these steps are united in the last picture. An agent labeled with a W has to wait in the current round.

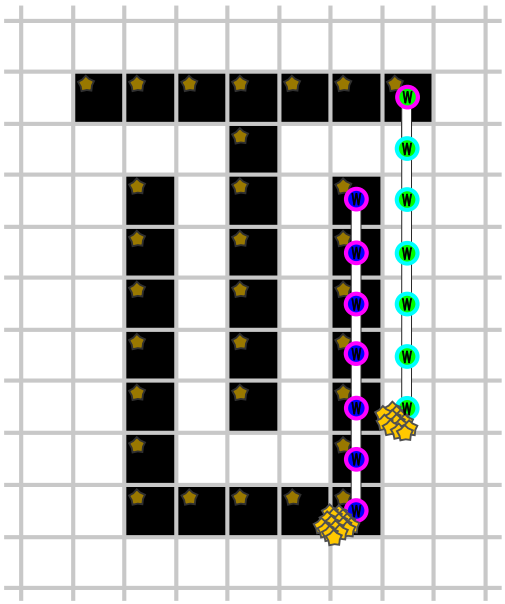


Figure 3.13: Example of Figure 3.9 continued. The pink color of the agents boundary indicates h_a = false, h_a = true is indicated in turquoise. Yellow stars indicate pixel weight, they are drawn pale if the pixel's weight has already been consumed. For clarity reasons the consumed weight is not forwarded during the run of the algorithm, these steps are united in this last picture, after the algorithms actual run. An agent labeled with a W has to wait in the current round.

Agents waiting for other agents to arrive—due to Rule 5b/c—may result in agents standing in lines of slope 1, as in the right tower in iteration #15. Again, agents only merge when they know that they belong to the same object (#38/#39). Hence, the chains for different objects can cross without interfering with each other (#173). In the end, we can observe a a vertical line for each object. Again, the collected weight can be picked up from the bottommost pixel of each vertical line.

Three-dimensional Objects and Grids. Note that our algorithm can easily be extended to cope with three-dimensional objects and pixel grids: We simply consider the 26 direct neighbors, for a sweep running along the y-axis, partners are searched for in adjacent pixels in the same columns along the x- and z-axis, respectively. Weight is passed on two partners that are below a pixels p according either to the x- or the z-direction (ties breaking arbitrarily), and so on. A plane of agents comprises the sweep, rather than a chain of agents for the two-dimensional case.

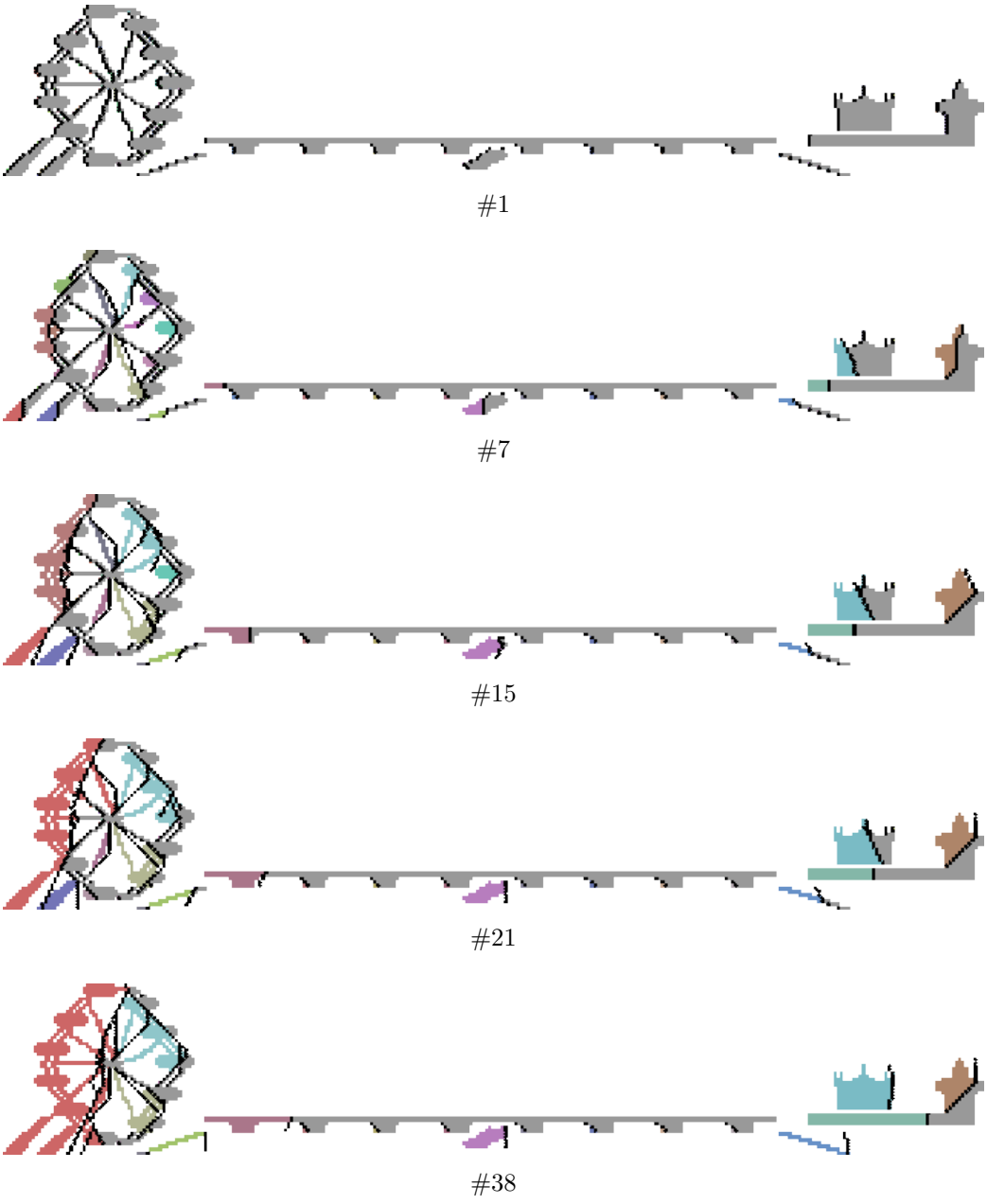


Figure 3.14: Another example for algorithm 3.1. Agents are shown in black, objects in grey. Pixels whose weight has already been consumed are colored. The color refers to the chain of partnership agents that carries this weight. Whenever new partnerships are established, the colors are also merged.

#39

#44

#115

#173

#342

#343

Figure 3.15: Example of Figure 3.14 continued.

Implementation on active pixels. After describing the algorithm we have a brief look at its implementation on a smart pixel grid. Each pixel maintains a list of the agents that are located on it. The pixel computes the decisions for all these agents. A synchronization with neighbor pixels is only necessary after Step 4 and Step 6. As these steps alter the variables we assumed to be known for agents in the 8-neighborhood. The other four steps are based on already available information only. For Step 1 no information about other pixels is necessary. The variables C_a and h_a have not been updated in Step 1, thus Step 2 can be performed without any synchronization with p's neighbors. Step 3 simply requires to send a message to corresponding partners, that is, to a neighbor pixel. Again, the decision in Step 5 is based on the already available knowledge of positions of partner agents.

3.4 Analysis of the Algorithm

We start this section with proving the correctness of the algorithm. In particular, the algorithm is able to distinguish interwoven objects. Thereafter, we analyze the runtime complexity of our algorithm. We will then analyze the number of agents that may be located on a single pixel at the same time. Finally, we have a brief look at the algorithms applicability.

3.4.1 Correctness

Theorem 1 (Correctness)**.** *When the algorithm stops, the weight of all pixels belonging to the same object is accumulated in a single agent. The agents for this object form a vertical line.*

Proof. As a first step we consider the establishment of partnerships. Initially, $C_a = \varnothing \ \forall a \in A$, that is, no partnerships have formed. An agent a decides on a new partner a' in Step 2 only if both are still in the object they started in—and never left it. Moreover, only agents with a common partner merge in Step 4. In addition, Step 5, conditions (b) and (c), make sure that agents never move until they have found partners in both adjacent rows if needed, that is, in case the adjacent pixels are object pixels. Consequently, agents that were created in different objects will never influence on each other.

So, we only need to consider agents belonging to one object. We consider a point in time when these agents do not form a vertical line. All agents in the left-most column have found their partners now. Thus, either one agent has an object pixel in $\mathsf{E}(p)$, and he will move. Or there is an agent with a partner to the right (i.e., in $\mathsf{NE}(p)$ or $\mathsf{SE}(p)$), and this agent will move.

Consequently, in the end all agents for the object form a vertical line.

In case they did not yet establish partnerships, they will now partner. Resulting in a line fully connected by partnership relations. The agents pass all weight of the object pixels down to the bottom-most agent (Step 3). □

3.4.2 Runtime Complexity

We proved that agents from different objects never influence each other. Hence, we can restrict the analysis to the case of a single object $\mathcal{X}$. We assume $W \times H$ to be the size of the smallest axis-parallel box that contains $\mathcal{X}$. For the proof of the runtime bound we look at objects with increasing complexity. We start our proof with *paths*, i.e., objects where each occupied pixel has at most two neighboring object pixels. On a pixel grid we may describe these paths by alternating vertical and horizontal lines. We denote the length of the vertical and horizontal parts by $H_i + 1$ (height) and $W_i + 1$ (width). We associate a positive sign with a downwards vertical and a rightwards horizontal line. The path is $(H_1, W_1, H_2, W_2, \ldots, H_V, W_V)$. To be precise, we start with a single pixel. To it, we attach $|H_1|$ pixels downwards (if $H_1 > 0$) or upwards ($H_1 < 0$). Then we attach $|W_1|$ pixels to the path's end, horizontally to the right ($W_1 > 0$) or left ($W_1 < 0$). We repeat this process for H_2, W_2, …, H_V, W_V.

We focus on the movement of the agents in this subsection. Thus, the weights of the pixels are not shown in the figures. Only the agents and partnership relations are shown (we do not distinguish different states of the agents by color).

Theorem 2 (L-shaped paths)**.** *Let $\mathcal{X}$ be an L-shaped path (H_1, W_1); see Figure 3.16. The agent on the horizontal line moves with speed $1/2$, it takes additional time of H to gain a vertical line at the rightmost end.*

Proof. We have $H_1 + 1 = H, W_1 + 1 = W$. Let a_b be the agent initially located at the bend of the L, see Figure 3.16. Let the others be $a_1, \ldots, a_k$ (named starting from the agent initially located above a_b). All agents create partnership edges, i.e., $a_b \in C_{a_1}, a_1 \in C_{a_2}, \ldots, a_{k-1} \in C_{a_k}$. In the first iteration a_b has no partner to the left, a partner above and no object below and in moving to the right a_b will neither be away from its home object nor exceed an L_∞-distance of 1 to a_1. Thus, a_b moves in the first iteration. The others do not move, as they would be leaving their home object and have no partners in $\mathsf{NE}(p)$ or $\mathsf{SE}(p)$ (see Rule 5d). In the second iteration a_b has to wait as its partner a_1 is to the left ($\mathsf{NW}(p)$). With a_b being in $\mathsf{SE}(p)$ for a_1, a_1 is allowed to go now. The other a_i's still wait. So, in the third iteration a_b is again allowed to move, as is a_2 (with a_1 in $\mathsf{SE}(p)$), all other agents wait.

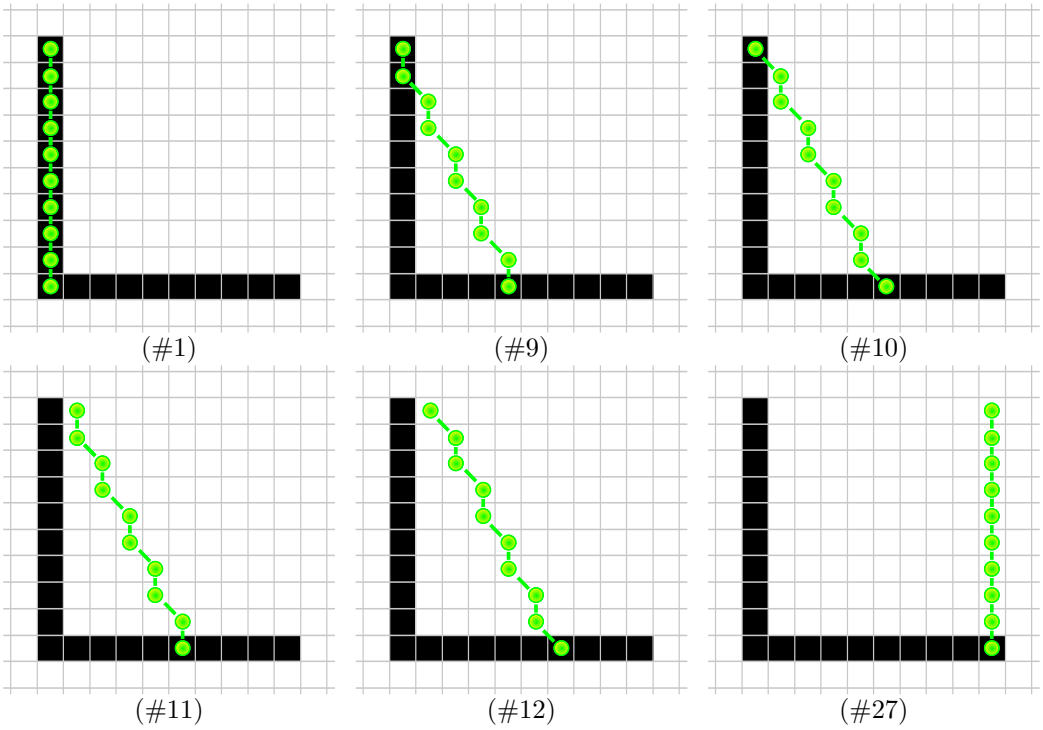


Figure 3.16: An L-shaped path and the algorithm sweeping over it. ($H_1 = 9, W_1 = 9$.)

This process goes on, a_b moves in every second iteration, i.e., with speed 1/2, and the other pixels form a rearwards line with slope 2 (at any point in time two agents occupy pixels in the same column). Thus, a_b reaches the right end after $2W - 1$ time units. As all the other agents are also moving with speed 1/2 it takes additional $H - 1$ time units until the vertical line at the right end is reached. □

Corollary 1. *Theorem 2 holds for an L-shaped path reflected along the x-axis.*

Theorem 3 (Windy paths with uniform height)**.** *Let $\mathcal{X}$ be an x-monotone path with $W_i > 0, \forall i$, $|H_i| + 1 = H, \forall i$, and alternating signs of the H_i's, see Figure 3.17. It takes at most $2W + H$ iterations to build up the right vertical line.*

Proof. By induction on the number of vertical lines, V.
The case $V = 1$ is covered by Theorem 2. Assuming the theorem is correct for V, we now show that it also holds for $V + 1$:

$\mathcal{X}$ can be covered by overlapping L-shaped paths, $(H_1, W_1), (H_2, W_2), \ldots,$ (H_{V+1}, W_{V+1}). Agents located in a pixel above or below such an overlap

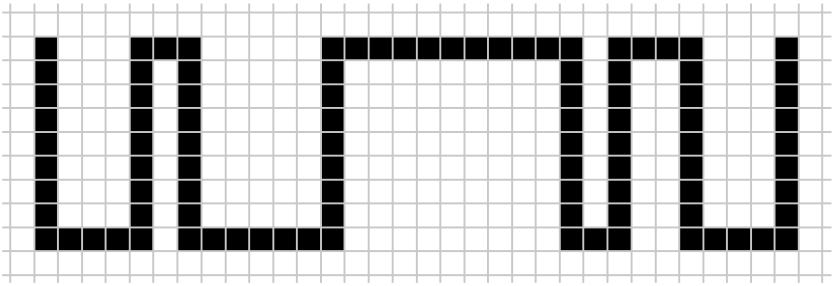


Figure 3.17: A windy path with all vertical lines of same height. ($|H_i| = 8\ \forall i, W_1 = 4, W_2 = 2, W_3 = 6, W_4 = 10, W_5 = 2, W_6 = 3$ and $W_7 = 4$.)

have to wait for an agent to arrive from the left before they can move, due to Rules 5b and 5c. If the waiting time is long enough ($2H - 4$) this results in agents standing in a line with slope 1 (see Figure 3.18). Whenever the agents from the left arrive, these lines of merged agents are allowed to move and do so with speed 1/2 (as there is always one agent located on a horizontal line, allowed to go whenever its partner caught up, thereby dragging the rest).

Now let us consider the situation in the first piece (H_1, W_1). No agent initially located on the vertical line added at the left has to wait for some agents approaching from the left. Thus, the agent initially located at the bend of the L, see Theorem 2, walks with speed 1/2 towards the next vertical line (arriving at iteration $2W_1 - 1$). The agents from the first vertical line approach the second in a line of slope 2, after iteration $2W_1 - 1$, as seen by the following two cases:

1. $W_1 < (H - 1)/2$: Some agents are still located on the first vertical line. When the first agent reaches the second vertical line the agent with which it will merge is not yet allowed to move, see the #5 in Figure 3.19. The slants move (horizontally) through the L-shaped parts, as seen in #6 to #9 of Figure 3.19.

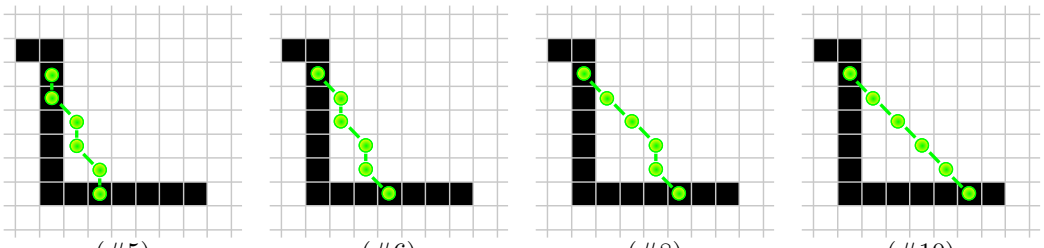


Figure 3.18: Agents initially located on an L-shaped path with a waiting agent form a line of slope at least 1.

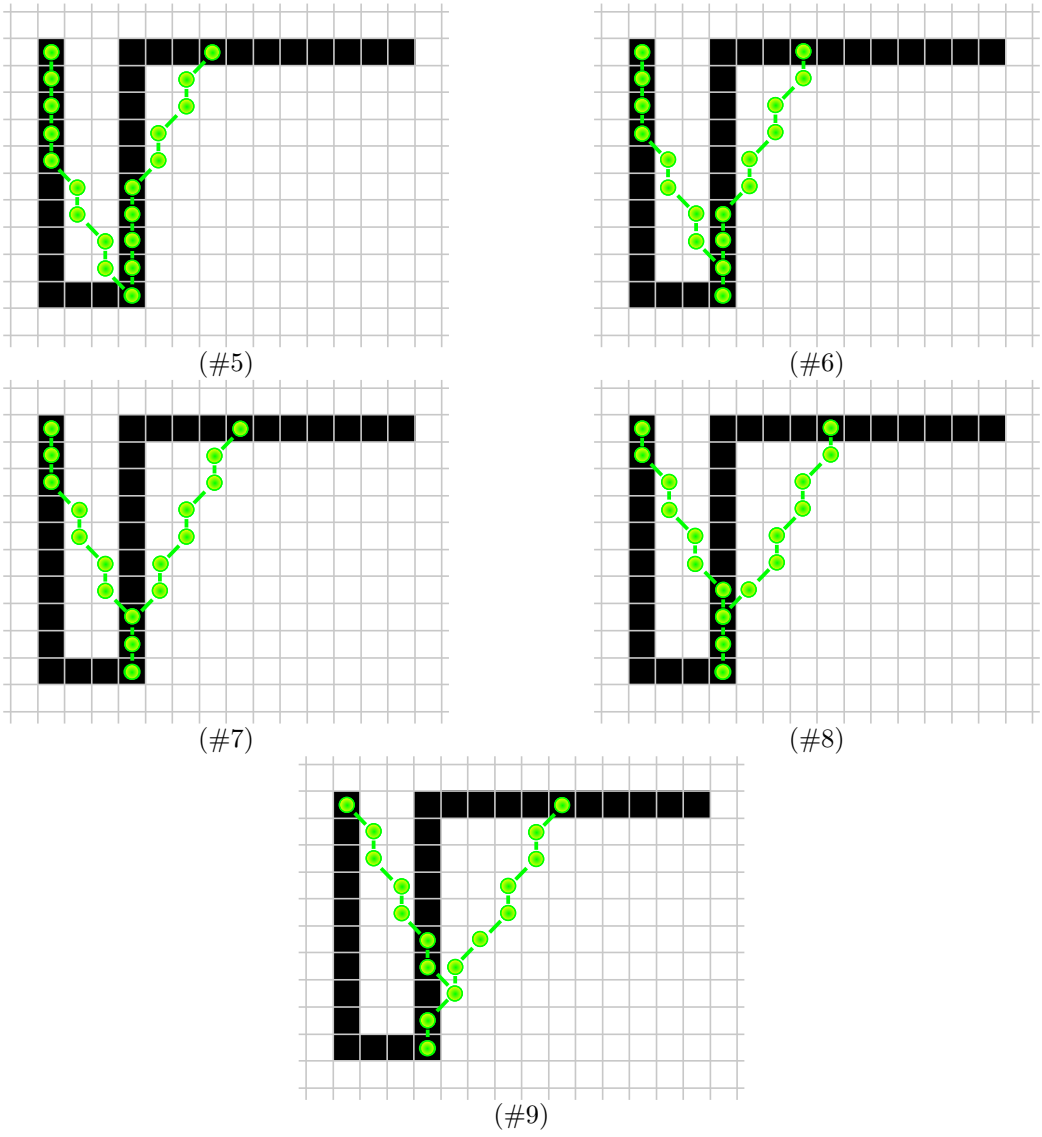


Figure 3.19: Case $W_1 < (H-1)/2$ in the proof of Theorem 3. ($W_1 = 3, H = 10$.)

At some point the merged agents are allowed to move. This happens after H_1 iterations. Thus, now all agents of the first vertical line keep on moving until they reach the right-most line. And, finally, a time of $H-(H_1-2W_1)$ is needed to gain the vertical line. Thus, altogether we have $H_1+2(W_V-1)+H-(H_1-2W_1) \leq 2W_V+H+2W_1 = 2W+H$ iterations.

2. $W_1 \geq (H-1)/2$: It may take additional time of $H^- \leq H$ until the merged agents are allowed to move. The agents keep on mov-

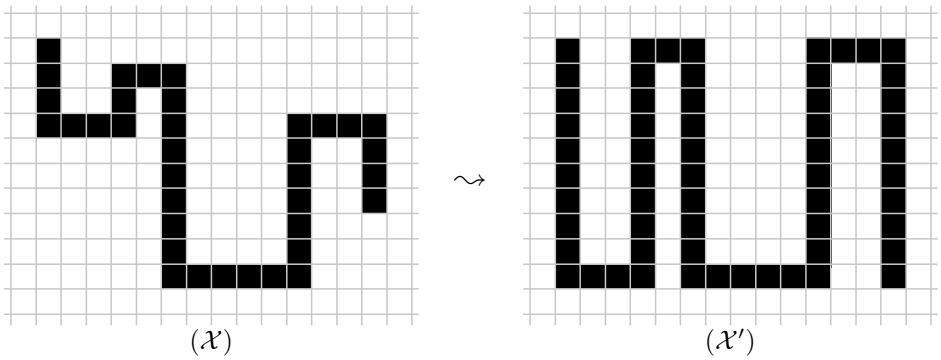


Figure 3.20: Windy path $\mathcal{X}$ and reduction $\mathcal{X}'$ to the previous case.

ing, collecting (by merging) all the others (time $2W_V$) and finally a time is needed to gain the vertical line. For $H^- = 0$ we have: $2W_1 - 1 + 2W_V + H \leq 2W_{V+1} + H = 2W + H$. For a positive H^-, we do not need an additional time of H to gain the vertical line, but $(H - \max\{H^-, H - H^-\} - 1)$. Consequently, for:

- $H^- > H - H^-$ we need $2W_1 - 1 + H^- + 2(W_V - 1) + (H - (H^- - 1)) \leq 2W_{V+1} + H = 2W + H$ iterations, and for
- $H^- \leq H - H^- (\Leftrightarrow 2H^- \leq H)$ we need $2W_1 - 1 + H^- + 2(W_V - 1) + (H - (H - H^-) - 1) \leq 2W_{V+1} + 2H^- \leq 2W + H$ iterations.

□

Theorem 4 (Windy paths of arbitrary height)**.** *Let $\mathcal{X}$ be an x-monotone path with $W_i > 0 \ \ \forall i$ and alternating signs of the H_i's, see Figure 3.20. It takes at most $2W + H$ iterations to build up the right vertical line.*

Proof. To prove the runtime we modify the object $\mathcal{X}$, and show that our algorithm is not faster on the modified object $\mathcal{X}'$, yet still runs in at most $2W + H$ on $\mathcal{X}'$.

The object $\mathcal{X}'$ consists of all vertical lines of $\mathcal{X}$ extended to the length of H and the horizontal lines shifted to the lowest respectively highest position in the bounding box, see Figure 3.20. We claim that our algorithm is not faster on $\mathcal{X}'$ than on $\mathcal{X}$. Then, the claimed runtime of $2W + H$ results from Theorem 3. We prove the claim in three steps:

Claim 1: *The time the agents need to reach the second vertical line from the first does only depend on W_1.* The agents starting from the first vertical line need not wait for agents approaching from the left. Theorem 2 shows

that the first agent reaches the next vertical line after $2W_1 - 1$, independent on the height H_1.

Claim 2: *For agents initially located on vertical lines other than the first an increased H_i may cause some agents to be able to move further to the right. Nevertheless, at least one agent stays on the line and is not picked up earlier.* The first part follows from the fact that a bigger H_i, determining the "leash length" of the agent moving on the horizontal line (with the waiting pixel located above or below an overlap of L-shaped parts holding the leash), gives this moving agent more freedom. The other agent still has to wait (see Figure 3.18).

As the agents of the first vertical line keep on moving with speed 1/2 it is picked up "in time" (and when moving to the right a line with slope at least 2 is reestablished).

Claim 3: *For a smaller height, H_i, the merging point enabling the agents initially located on the next vertical line to move, is not reached later. That is, with a bigger H_i the waiting Rule 5d is not resolved earlier.* We need to distinguish two cases:

- In case the first vertical line is longer than (or equal to) the second ($|H_i| \geq |H_{i+1}|$), the time until merged agents are allowed to move depends on the height: the slant moves (horizontally) through the L-shaped parts—this takes longer for a bigger height.

- In case the first vertical line is shorter than the second ($|H_i| < |H_{i+1}|$), the "new" agents added parallel to agents from the $(i+1)$th vertical line may overstep the second vertical line before some agents initially located in the second vertical line are enabled to move. Nevertheless, the pulling agent is still one of the second vertical line and consequently not to the left of the "new" ones. The slant keeps on moving horizontally through the L-shaped path, enabling the agents of the two vertical lines to move at the same time as in $\mathcal{X}$. □

Theorem 5 (X-monotone paths)**.** *Let $\mathcal{X}$ be an X-monotone path with $W_i > 0$, $\forall i$, see Figure 3.21. It takes at most time $2W + H$ to build up the right vertical line.*

Proof. Again, we construct an object $\mathcal{X}'$ and show that the algorithm is not faster on $\mathcal{X}'$ than on $\mathcal{X}$. Consider the sign of the vertical lines (i.e., whether the next horizontal line is situated below or above the last one). Swing out all stairs with a row of vertical lines of the same sign, see Figure 3.21. Hence, $\mathcal{X}'$ is an object of the type considered in Theorem 4, and the running time is at most $2W + H$.

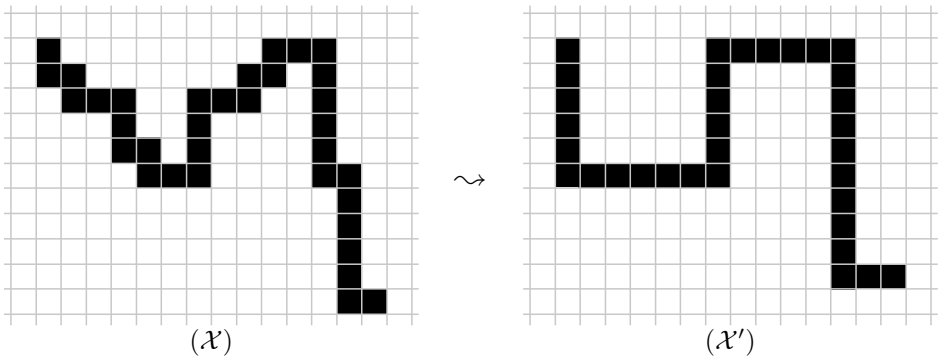


Figure 3.21: X-monotone path $\mathcal{X}$ and its reduction $\mathcal{X}'$.

We claim that merged agents are not allowed to move earlier on $\mathcal{X}'$. Let the vertical lines of the first staircase be $\ell_1, \ldots, \ell_r$, from left to right. We compare it to the corresponding L-shaped path (with height $(\sum_{i=1}^{r} H_i) + 1$ and width $(\sum_{i=1}^{r} W_i) + 1$).

When the agents of $\mathcal{X}'$ gain a line of slope 2 for the first time, the agents from $\ell_1, \ldots, \ell_r$ are at most in lines with slope 2 (as they are shorter), some agents may have gone further to the right, resulting in lines with slopes not less than 1.

The agents of ℓ_1 reach the merge point after $2W_{\ell_1} - 1$. At this time we have a parallel line in $\mathcal{X}'$ with all agents to the left of the ones from $\ell_1, \ldots, \ell_r$. □

Theorem 6 (Arbitrary paths)**.** *Let $\mathcal{X}$ be a path, see Figure 3.22. It takes at most $2W + H$ iterations to build up the right vertical line.*

Proof. We construct a set $\mathcal{X}'_1, \ldots, \mathcal{X}'_z$ of objects (paths) by splitting the vertical lines of $\mathcal{X}$ at the rightmost position of the bounding box, if these lines exist, and projecting the maximal height H to the left of each part. Note that this may result in circles. Then, we take the longest of the paths $\mathcal{X}'_1, \ldots, \mathcal{X}'_z$ (in terms of time), let this path with the projection on the left be $\mathcal{X}'$. We

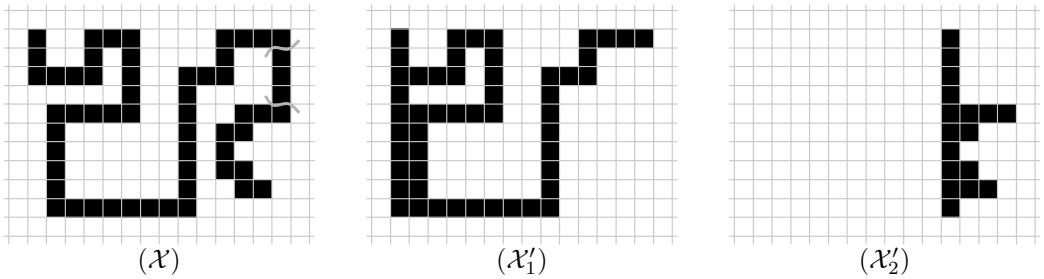


Figure 3.22: Arbitrary path $\mathcal{X}$ being transformed into simpler pieces $\mathcal{X}'_1$ and $\mathcal{X}'_2$.

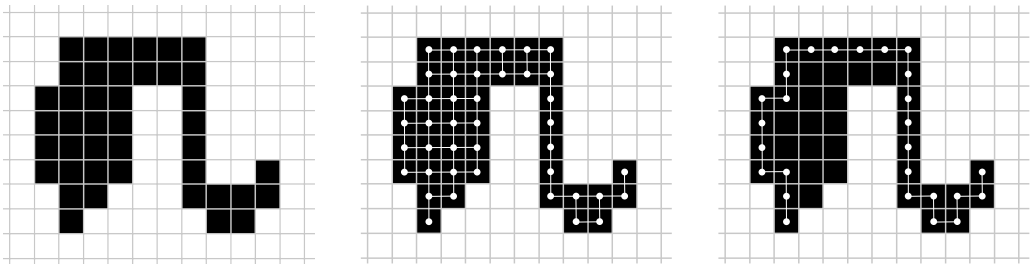


Figure 3.23: Reducing the general case to analyzing the most complex subpath.

claim that $\mathcal{X}'$ is not processed faster than $\mathcal{X}$. We prove the claim in two steps:

Claim 1: *The height stays the same.* Obvious, as we projected the maximal height to the left of each part, in particular to the left of $\mathcal{X}'$.

Claim 2: *The new created agents will not pick up agents to the right earlier than these will be enabled to move in* $\mathcal{X}$. We add a vertical line, i.e., the agents have to be pulled by an agent (or more) on an existing horizontal line. This agent keeps its speed and is never to the left of the added agents before they potentially merge with "old" agents: When they merge with "old" agents the merged agents inherit the conditions for both.

Hence, if they had to wait they still do, but as waiting for agents from the left they will be enabled to move.

Thus, the new added agents on the first (leftmost) vertical line will not influence on the time "old" agents from the leftmost end need to reach the right most end ($2W + H^*, H^* \leq H$). They are at least pulled and may then build a vertical line (time $H - H^*$), resulting in running time of at most $2W + H$.

□

Theorem 7 (General case)**.** *Let* $\mathcal{X}$ *be the given object. The running time of the algorithm is at most* $2W + H$*.*

Proof. We may consider $\mathcal{X}$ as a graph $G_{\mathcal{X}}$: $\mathcal{X}$ occupies pixels, the nodes, and neighboring (4-neighborhood) pixels are adjacent, see Figure 3.23. We prove the running time in four steps:

Claim 1: *Walking inside an object is faster than walking outside of an object.* Obviously, as outside of an object movement is further restricted by Rule 5d.

Claim 2: *Reducing* $G_{\mathcal{X}}$ *to its leftmost edges whenever vertical edges are parallel (and deleting the horizontal edges in these areas, preserving connect-*

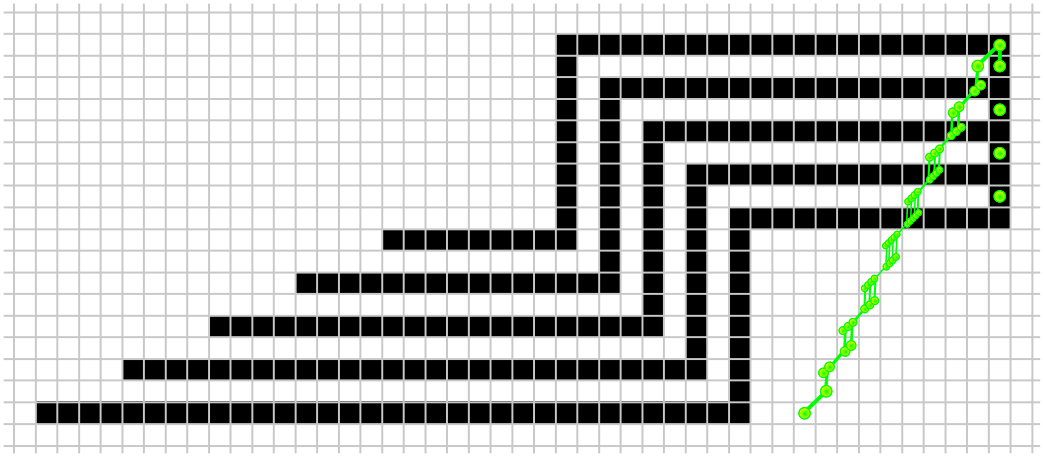


Figure 3.24: Worst-case on agents per pixel.

ing edges) we gain a tree, $T_{\mathcal{X}}$. Obviously, as the construction does not leave any cycles.

Claim 3: *The algorithm does not run faster on $T_{\mathcal{X}}$ than on $G_{\mathcal{X}}$.* Results from (1) and (2).

Claim 4: *The running time on $T_{\mathcal{X}}$ is at most $2W + H$.* In $T_{\mathcal{X}}$ we consider all paths that end on the right. Then, we take the longest (in terms of time) of these, let it be P (decide on P from left to right at branching points).

Analogously to the proceeding in the proof of Theorem 6 we project the maximal height H to the left of P.

We do not shorten the running time of "old" agents on P from the left to the right. Arguments like in the proof of Theorem 6 yield the $2W + H$. □

3.4.3 Agents per Pixel

A pixel p stores all agents (with corresponding variables) that are currently located on p. The pixel performs the computations and communication for all of the agents stored on it.

As described in Section 3.1 we assume the $\mathcal{LOCAL}$ model of Peleg [Pel00], in which a pixel may perform all computations and communication for all agents that currently reside on it in a single communication round.

Figure 3.24 gives a worst-case example for the number of agents per pixel for our algorithm. We can enlarge this example, yielding pixels with $\Omega(W)$ agents stored on it at the same time. As agents only move horizontally and agents will initially never be located on horizontally adjacent pixels (we only generate agents on object pixels with a non-object left neighbor), there is a trivial upper bound of $\lceil W/2 \rceil$ on the maximal number of agents on a pixel.

For any realistic input we did not observe more than three agents located on the same pixel, thus, we consider this possibly large number of agents residing on a pixel at the same time irrelevant for practical applications.

If, on the other hand, we consider the more restricted $\mathcal{CONGEST}$ model that limits the message size per communication round to $O(\log n)$, our algorithm has a runtime complexity of $O(W(W+H))$. Nevertheless, we estimate that this model does not reflect the hardware platform for which we developed the algorithm: It was designed so as to allow for a transfer of every cell's memory to a neighboring cell in a single round. Thus, a limitation of the message size would be rather artificial.

3.4.4 Implementation

In Section 3.4.2 we proved a runtime complexity of $O(W + H)$ for our algorithm. Thus, in theory it is highly efficient. To verify whether it is also applicable, it was implemented for actual hardware in VHDL. An essential element of this implementation was the transfer of our model into hardware: Each pixel became a processing element (PE) on the final chip. The control unit of each PE was implemented as a finite state machine.

An important aspect for the algorithms implementation was the number of agents simultaneously residing on one pixel. As mentioned in Section 3.4.3 as many as $\Theta(W)$ agents may be located on a pixel at the same time. However, only a small constant number of agents per pixel can be observed for any realistic input. In the given implementation we allowed for three agents to occupy a single pixel at a time.

The algorithm was first simulated. Thereafter, the design was synthesized for a field-programmable gate array (FPGA) and for an application-specific integrated circuit (ASIC). (Both, FPGA and ASIC, are integrated circuits. While an FPGA can be (re-)configured after manufacturing, thus, allowing for many different applications, an ASIC is a chip developed for specific applications during the last procedural process.)

The synthesis resulted in chips of reasonable size and low power consumption. So, with good cause we think that our algorithm can be turned into ASICs, also for industrial typical resolutions. The observed low power consumption and high chip frequency suggest that industrially manufactured chips could well compete with a classic, centralized design. See [FFK+09] for details of the implementation.

3.5 Conclusion

We presented a novel sweepline algorithm for computing attributes, as the center of gravity, for objects in a smart pixel image. The attributes are extracted via a Principal Component Analysis. All steps in the algorithm are local. We use an agent sweep for every object present in the image, even though the start of the sweeping agent chains is purely local, joining multiple chains for an object during the run of our algorithm if necessary.

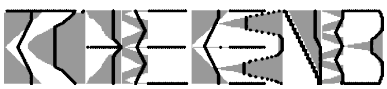
The time complexity of our distributed algorithm is $O(W+H)$ for objects within a bounding box of $W \times H$ pixels. Thus, it outperforms any centralized or in-object distributed algorithm—as $\Omega(WH)$ pixels may have to be treated. Our algorithm can cope with any number of arbitrary intertwined objects.

This shows that using mobile agents (with limited capabilities) for a geometric problem that does not necessarily asks for these agents can result in a faster algorithm.

Based on these theoretical results, we showed that our algorithm is simple enough to be turned into hardware by synthesizing it for FPGA and ASIC.

As mentioned, our algorithm can easily be extended to cope with three-dimensional objects and pixel grids. We are confident that the running time for the three-dimensional case is linear in $W + H + D$ (W being the width, H the height and D the depth of the smallest axis-parallel box that contains the considered object). Of course, a more detailed analysis of the three-dimensional case could be the subject of future work.

Currently, several agents can be located on a single pixel. For real-world data this does not seem to be an issue. And for our application we could get rid of this problem by simply using a picture taken after a few milliseconds and retry the analysis. By then the conveyor belt will have moved slightly. Nevertheless, improving the theory in this aspect would be beneficial. An idea would be to shrink the agent chains from inactive ends, probably resulting in an increased run time complexity (Kröller [Krö10]).



CHAPTER 4

Exploration with a Myopic Watchman with Discrete Vision

In this chapter we study the problem of perceiving an entire polygon with a mobile agent that has a limited range of visibility and may only acquire visibility information (take a scan) when it is stationary. As both, moving and scanning, takes time, our cost function is a linear combination of tour length and number of scans. We seek an agent's tour that minimizes the cost.

4.1 Problem Description

Finding a route inside of a polygonal region such that each point of the polygon becomes visible at least once for a watchman walking along this route is a classical problem in computational geometry, the *watchman route problem* first introduced by Chin and Ntafos [CN86].

Thinking of a mobile robot rather than an "ideal" watchman gives rise to the following question: What happens if the watchman does not have continuous vision with an unlimited range of visibility?

In case of a robot equipped with a 3D laser scanner both restrictions apply. The distance in between two laser beams meeting an object is so large for objects in a certain distance, that the scan does not allow for a proper perception of far-off objects. Hence, rather than operating with an unlimited range of visibility, the robot has a limited scan range. Moreover, the single scans (the point clouds resulting from the scans) taken by the robot

have to be mapped after the scan procedure. To do so properly, the robot should be immobile while taking the scan. Resulting in proper defined point clouds. Every scan takes a certain amount of time. Thus, perception of the environment is possible only at discrete points in the environment; the robot does not have continuous vision.

The abstract image of this kind of visibility are balls of a certain radius around points, the visibility range of scans taken at these points. Whether these balls can be depicted as squares or disks depends on the metric we apply (L_∞- or L_2-metric, respectively). Consequently, the request of making every point of the polygon visible to the watchman is equivalent to covering the polygon with these balls (with an additional constraint that guarantees visibility, that is, the line of sight to all points must lie inside of the polygon).

Finding a small cover of disks or squares is a classical problem (see Tóth [Tót49], Aupperle et al.[ACKO88], Keil [Kei86], Franzblau and Kleitman [FK84] and Wu and Sahni [WS90]) with many applications (see for example Tanimoto and Fowler [TF80]). The same is true for the problem of finding a shortest tour that connects a given set of locations (see Mitchell [Mit00] for a survey on the Traveling Salesman Problem (TSP) and its variants). In fact the problem of a watchman with range limited discrete vision combines both problems. We intend to find a small set of scan points that covers the environment and a short tour that visits all these points. We call this resulting problem the *Myopic Watchman Problem with Discrete Vision* (MWPDV).

4.1.1 Problem Statement

We are given a polygon P. In general, P may be a polygon with holes; in Sections 4.2, 4.3 and 4.4, P is an axis-parallel polygon with integer coordinates, a *rectilinear grid polygon*.

Our robot, R, has discrete vision, i.e., it can perceive its environment when it stops at a point and performs a scan, which takes c time units. The points where R stops are called *scan points*. From a scan point p, only a ball of radius r is visible to R, either in L_∞- or L_2-metric. A set $\mathcal{S}$ of scan points *covers* the polygon P, if and only if for each point $q \in P$ there exists a scan point $p \in \mathcal{S}$ such that q sees p (i.e., $qp \subset P$) and $|qp| \leq r$.

We then define the *Myopic Watchman Problem with Discrete Vision* (MWPDV) as follows: Our goal is to find a tour T and a set of scan points $\mathcal{S}(T)$ that covers P, such that the total travel and scan time is optimal, i.e., we minimize $t(T) = c \cdot |\mathcal{S}(T)| + L(T)$, where $L(T)$ is the length of tour T. Instead of minimizing this linear combination of scan point number and tour length, we may also consider the bicriteria problem. That is, we want to give

a solution (in fact an approximation) for both the scan number and the tour length.

We distinguish MWPDV milling and MWPDV lawn mowing. In the milling case the robot must stay in the polygon all the time, that is, all scan points and the tour must lie inside of P. For the lawn mowing variant the robot is allowed to leave the polygon, which in this case can be disconnected.

In the following Sections we will denote the MWPDV variant we currently consider by choosing one of the possible attributes in each case: rectilinear/general MWPDV milling/lawn mowing for rectangular/circular scan range. Rectilinear MWPDV denotes that we restrict to rectilinear grid polygons. In fact, we describe results for the milling variant, a PTAS for a lawn mowing variant was given by Mitchell [FMS09]. We use the terms *visibility range* and *scan range* synonymously.

Organization of this Chapter. In the remainder of the current section we present related work and we give some annotations on the MWPDV. In Section 4.2 we prove NP-hardness results. Thus, we focus on approximation algorithms. For different variants of the MWPDV (milling) we present approximation algorithms in the subsequent sections: In Section 4.3 we consider rectilinear grid polygons and unit L_∞ scan range and give a 2.5-approximation algorithm. This approximation generalizes the 2.5-approximation for continuous milling by Arkin et al.[AFM00]. In Section 4.4 we consider rectilinear grid polygons and unit L_2 scan range. For this problem variant we present a 4-approximation strategy. Section 4.5 provides an approximation algorithm for the case of general polygons and an L_2 scan range. The algorithm has an approximation factor of $\max(\frac{21}{4}, \frac{\pi r}{a} + \frac{\pi r}{2} + \frac{\pi}{2})$, a constant factor for a bounded ratio of r/a. All these approximation algorithms hold for both, the combined and the bicriteria version, as we account for the scans and the tour separately. We conclude in Section 4.6.

We published the results presented in this chapter in [FMS09] (and a journal version [FMS10]).

4.1.2 Related Work

Focusing on the discrete visibility only, we face the related classical *art gallery problem*, that aims at locating a minimal set of scan points. On the other hand, considering the tour planning aspect results in the *watchman route problem*, that asks for a shortest route along which every point of the polygon becomes visible at least once (for an explorer with continuous, unrestricted vision). In addition, visiting all points of a given set is the classical *Traveling Salesman Problem* (TSP). Finally, the tour planing with a restricted visibility

range—without accounting for discrete visibility—is closely related to *lawn mowing* and *milling problems*. For a cutter of a given size and shape the milling problem asks for a shortest tour of the cutter, such that the entire polygon is covered and such that the cutter is located inside of the polygon for the entire route.

Art Gallery Problem. Looking for a set of discrete positions from which all points of the polygon become visible for (stationary) guards (with unlimited visibility) located at these points is the *art gallery problem* (see Urrutia [Urr00], O'Rourke [O'R87] and Shermer [She92]). This problem is based on a question posed by Klee, who asked for the number of guards necessary to be placed in an art gallery to guarantee complete visibility coverage of all exhibits. That is, one asks for the minimum number of immobile guards, $G(P)$, that cover all of a polygon P. $g(n)$ is the maximum $G(P)$ over all polygons of n vertices. Two points in a polygon are defined to be *visible* to each other if the line connecting the points lies inside of P. Chvátal [Chv74] was able to show that $\lfloor \frac{n}{3} \rfloor$ (stationary) guards are occasionally necessary and always sufficient so that every point of a polygon with n vertices is visible from at least one guard position ($g(n) \leq \lfloor \frac{n}{3} \rfloor$). Fisk [Fis78] gave a short and simple proof for Chvátal's result. He triangulated the polygon and 3-colored the resulting structure, i.e., assigned a color (out of 3) to every vertex such that two adjacent vertices do not share a color. The least frequently used color cannot be used more than $\frac{n}{3}$ times (the sum of all colored vertices would otherwise be greater than n). Consequently, by choosing the set of vertices assigned to the least frequently used color Fisk obtained less than $\frac{n}{3}$ guards. As each triangle of the triangulation must have all three colors at its vertices, the chosen set sees all triangles and thus the entire polygon. $g(n)$ was also considered for more restricted classes of polygons, e.g., Kahn et al. [KKK83] established $g(n) \leq \lfloor \frac{n}{4} \rfloor$ for orthogonal polygons.

Beside various classes of polygons to be guarded (orthogonal polygons, polygons with or without holes etc.), variations on the abilities of the guards have been examined. In the classical Art Gallery Problem the guards are stationary, i.e., bound to a single point that may be located at any possible position inside of P; these are called *point guards*. If the feasible locations are restricted to vertices, we deal with *vertex guards*. Moreover, guards may have the ability to move along certain structures. *Edge guards* are allowed to move along an edge and survey all points visible to some point on this edge. Instead of patrolling along an edge, *diagonal guards* move along diagonals, *mobile guards* are allowed to use both. See Shermer [She92] for these definitions.

Alternatively, variations on the guards' task have been considered, for example, Laurentini [Lau99] required visibility coverage for the polygon's edges only.

While the presented work of Chvátal and its variants on art gallery theorems focused on a sufficient number of guards for all polygons of n vertices ($g(n)$), the art gallery problem asks for a minimum set of points (the guards, $G(P)$), such that every point in a given polygon P is visible from at least one point in $G(P)$.

O'Rourke and Supowit [OS83] proved this problem to be NP-hard by a reduction from 3SAT, for guards restricted to be located on vertices and polygons with holes. Lee and Lin [LL86] showed NP-hardness also for simple polygons. This result was extended to point guards (that are allowed to be located anywhere inside of P) by Aggarwal (see [O'R87]); Schuchardt and Hecker [SH95] gave NP-hardness proofs for rectilinear simple polygons, both for point and vertex guards.

Based on these NP-hardness results approximation algorithms gained interest. Eidenbenz et al. [ESW01] established lower bounds on the achievable approximation ratio. They gave a lower bound of $\Omega(\log n)$ times the optimal number of guards for polygons with holes. For simple polygons and vertex, edge and points guards they showed that there is a constant $\delta > 0$ such that no polynomial algorithm can achieve an approximation factor of $(1 + \delta)$. For restricted versions approximation algorithms have been presented. Efrat and Har-Peled [EHP06] gave a randomized approximation algorithm with logarithmic approximation ratio. Ghosh [Gho10] presented algorithms for vertex and edge guards only, with an approximation ratio of $O(\log n)$. The algorithm is based on considering the arrangement of the visibility polygons of P's vertices and using a set cover approach. For point guards Nilsson [Nil05] gave $O(OPT^2)$-approximation algorithms for monotone and simple rectilinear polygons. For the even more restricted case of 1.5D terrains Ben-Moshe et al. [BMKM05] present a constant-factor approximation. Another constant-factor approximation was provided by Clarkson and Varadarajan [CV05], King [Kin06] presented a 5-approximation, Elbassoni et al. [EKM+11] improved this factor and obtained a 4-approximation. Gibson et al. [GKKV09] presented a PTAS.

See Ghosh [Gho10] for an overview of approximation algorithms for art gallery problems.

Instead of aiming for approximations, algorithms for placing $g(n)$ guards that cover the entire polygon have been considered. Avis and Toussaint [AT81] gave an algorithm based on Fisk's triangulation proof of Chvátal's theorem to place $\lfloor \frac{n}{3} \rfloor$ guards in $O(n \log n)$. Due to Chazelle's linear time triangulation algorithm [Cha91] this could be improved to $O(n)$.

In addition, the problem of covering the polygon has been considered. As the visibility polygon of a single guard is star-shaped, covering the polygon with star-shaped polygons yields a set of guards. Keil [Kei86] gave an $O(n^2)$

algorithm for covering a horizontally convex orthogonal polygon with star-shaped polygons. That is, for this restricted class of polygons a minimum set of guards can be found in polynomial time. For other types of covers, e.g., covering with rectangles, the resulting number is of course not the optimal set of guards but may yield an approximation. For example, Franzblau and Kleitman [FK84] presented an algorithm to cover a rectilinear polygon with rectangles. However, many minimum cover problems are NP-hard. Culberson and Reckhow [CR88] presented a couple of NP-hardness results: amongst others for covering (even simple) polygons with convex polygons.

For some other special cases optimal solutions can be computed in polynomial time. Worman and Keil [WK07] presented an exact algorithm for rectilinear polygons an a special visibility restriction: Two points see each other if and only if the orthogonal rectangle defined by those points is fully included in the polygon (denoted by *r-visibility* in [WK07]).

Moreover, heuristic solutions have been considered. Amit et al. [AMP10] presented heuristic solutions, with the guards taken from different candidate sets, like vertices and more enhanced points. In addition, they presented some lower bounds that they used for a comparison with the heuristic solutions. The lower bounds are based on the visibility polygons of a certain set of points. Considering the visibility polygons of two points, we know that we need at least two guards to survey these points in case the visibility polygons are disjoint. Therefore, a candidate point set S is chosen and we build a graph with vertices for these points, plus edges in case of intersecting visibility polygons. Then we search for the maximum independent set. Because finding a maximum independent set is NP-hard, Amit et al. use a greedy strategy: iteratively add the node with the smallest degree to the set I, and remove this node and its neighbors from S. This greedy approximation algorithm has a performance guarantee of only $1/(\frac{|E|}{|V|}+1)$ (see Jansen and Margraf [JM08]). $|I|$ gives a lower bound on the number of guards necessary to cover S, and thus a lower bound on the guards needed for P. Amit et al. include convex vertices and midpoints of edges incident to two reflex vertices to the candidate sets. Couto et al. [CSR07, CdSdR08, CdRdS09] consider exact solutions for a special case of the Art Gallery Problem: vertex guards in simple polygons. Their approach is based on a set-cover integer program, applied to a grid discretization, which is iteratively refined if necessary. Recently, Baumgartner et al. [BFKS10] presented a primal-dual algorithm based on linear programming for the general art gallery problem that gives lower bounds on the number of guards and—in case of convergence and integrality—ends with an optimal solution.

Watchman Route Problem. The classical problem that asks for a short-

est route inside of a polygon P along which every point of P becomes visible to at least one point along the route, the watchman route problem, was stated by Chin and Ntafos [CN86]. They presented a strategy for a watchman route in simple rectilinear polygons in $O(n \log n)$. They defined *essential edges*, the strategy deletes all parts of the polygon outside of these edges and triangulates the remaining polygon. This polygon is unfolded along the essential edges. Finding the shortest watchman route in a simple rectilinear polygon is then equivalent to finding the shortest path between an edge and its image in the new geometrical structure. Moreover, Chin and Ntafos showed that the watchman route problem is NP-hard for polygons with holes (even if polygon and holes are convex or rectilinear) by a reduction of the geometric traveling salesman problem.

Later Chin and Ntafos [CN91] presented an algorithm for the computation of a shortest watchman route in a simple polygon in $O(n^4 \log \log n)$ in case of a given starting point on the boundary. Improvements of the runtime (and the correction of a small error) were presented by Tan et al. [THI91, THI99, TH93]. Those algorithms rely on the linear time triangulation algorithm by Chazelle [Cha91].

A first polynomial time algorithm for the case without a given starting point was presented by Carlsson et al. [CJN93]. (The starting point condition may result in an optimal watchman route that is arbitrarily longer than the optimal route without this restriction.) As this algorithm runs in $O(n^6)$, Nilsson [Nil01] gave a constant-factor approximation that runs in $O(n \log n)$.

The online version of the watchman route problem, in which the polygon is unknown in advance, has been studied extensively. Kleinberg [Kle94] gave a lower bound of 5/4 for the competitive ratio of any online algorithm in simple polygons without a specified starting point on the boundary, but in the polygon's interior. For simple rectilinear polygons with a given starting point on the polygon's boundary, Deng et al. [DKP98] presented the 2-competitive greedy-online strategy. Hammar et al. [HNS02] gave an improved version of this strategy, with a competitive ratio of 5/3. This ratio was further improved to 3/2 by Hammar et al. [HNP06]. For the less restricted case of simple polygons Hoffmann et al. [HIKK01] presented a 26.5-competitive strategy based on a structure called angle hull, thereby improving their bound of 133 given in [HIKK97].

In contrast to those competitive strategies, Albers et al. [AKS99] showed that no competitive online algorithm can be found for polygons with holes.

Watchmen with discrete vision. Discrete vision was considered by Carlsson et al. [CNN93]: For a robot with a vision system active only at discrete points, vision points, they asked for a shortest route connecting those point such that each point of the polygon is visible from at least one vision

point. They proved that finding the minimum number of vision points along a shortest watchman route is NP-hard in simple polygons (by a modified proof of Lee and Lin [LL86]).

Wang et al. [WKG07b] assigned cost to both traveling and scanning (at discrete points, view points, and with unlimited range of visibility). They considered only a discrete set of potential view points and present an approximation algorithm for this problem, the Traveling View Planning Problem (Traveling VPP), that uses an IP-formulation. Moreover, Wang et al. [WKG07a] considered a problem without a given discrete set of possible scan positions. They presented the Whole Edge Covering Generalized Watchman Route Problem (WEC-GWRP), that asks for a tour along which scan points are located such that the entire polygon boundary becomes visible and every edge of the polygon is entirely visible from one scan point. Wang et al. gave an approximation algorithm for the WEC-GWRP by sampling via a visibility cell decomposition and then using the approximation result for the Traveling VPP.

Arkin et al. [AAB+06] presented a polynomial time approximation scheme for the problem of covering a set of points by a number of scans and connecting these scans by a tour, the objective function being a linear combination of scan cost and travel cost. Here, the set to be scanned is discrete. Moreover, scan cost is a function of the scan radius, that is, the size of the scan radius may vary.

Online problems with discrete visions have also been considered. Ghosh et al. [BGS01, GBBS08] assigned cost for the scans only (that is, the route connecting the scan points does not infer any costs). Their online algorithm has a competitive ratio that depends on the number of reflex vertices, r. In addition to the discrete vision Ghosh et al. considered exploration with a robot with a limited visibility range R, with the same cost model. The presented algorithm has a competitive ratio of

$$\lfloor \frac{8\pi}{3} + \frac{\pi R \times Perimeter(P)}{Area(P)} + \frac{(r+h+1) \times \pi R^2}{Area(P)} \rfloor$$

for a polygon P with h holes.

For an online watchman problem with unrestricted but discrete vision, Fekete and Schmidt [FS10] presented a strategy with constant competitive ratio for polygons of bounded feature size and with the assumption that each edge of the polygon is fully visible from some scan point. For limited visibility range, Wagner et al. [WLB00] discussed an online strategy that chooses an arbitrarily uncovered point on the boundary of the visibility circle and backtracks if no such point exists. As cost they only considered the length of the path used between the scan points, scanning causes no cost. Then,

they can give an upper bound on the cost as a ratio of total area to cover and squared radius.

Another type of short-sighted explorer was considered by Icking et al. [IKKL05]. The robot moves in an unknown cellular environment under the assumption that a cell is explored when it is entered during the exploration.

See Ghosh and Klein [GK10] for a survey on online exploration algorithms with different capabilities of the explorer.

Lawn Mowing and Milling. The problem of covering a polygon by a mobile cutter moving along a shortest route (with the cutter inside of the polygon at all times), the milling problem, has been considered by Arkin et al. [AFM00]. They presented a 2.5-approximation algorithm for milling, running in $O(n \log n)$. Moreover, they gave a $(3 + \varepsilon)$-approximation method for the lawn mowing variant, in which the cutter is allowed to leave the polygon. A variant that asks for minimizing the cost of turns in a covering tour with a given tour was considered by Arkin et al. [ABD+01].

The online variant of this problem was considered by Gabriely and Rimon [GR03]. They restricted the cutter to be a square of size D that is allowed moves in directions orthogonal to the cutter's four sides only. For this version they present worst-case optimal online algorithms.

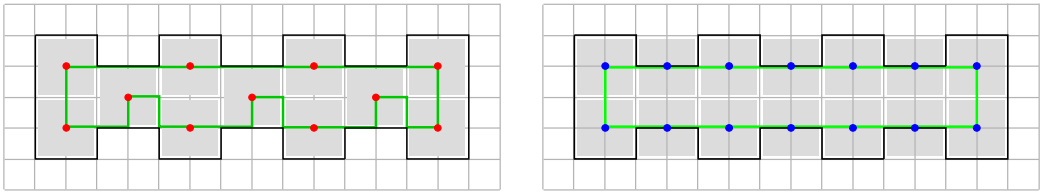
Traveling Salesman Problem. The traveling salesman problem asks for a shortest tour that visits all points of given (finite) set. Papadimitriou [Pap77] showed that this problem is NP-hard, even for point sets in the Euclidean plane. The TSP has been studied extensively, see for example Lawler et al. [LLRS85]. A couple of approximation algorithms has been presented for the TSP. For general metrics the 1.5-approximation by Christofides (see Schrijver [Sch03]) is the best known. Polynomial time approximation schemes were presented by Arora [Aro98] and Mitchell [Mit99] for point sets embedded in the Euclidean plane.

Finding a shortest tour that visits at least one point from each neighborhood is the TSP with neighborhoods (TSPN). Dumitrescu and Mitchell [DM03, Mit07] presented approximation algorithms and a polynomial time approximation scheme for the TSPN.

4.1.3 Some Notes about the Problem

Before we start to present our results on the MWPDV, we like to highlight two important aspects for any further consideration.

After observing the NP-hardness of the MWPDV (proofs in Section 4.2), we are interested in approximation algorithms or even in polynomial-time approximation schemes (PTAS). As mentioned in the introduction to this section, the MWPDV combines the minimum cover problem and the TSP, in



(a) (b)

Figure 4.1: Example showing the impossibility of combining a minimum cover and a TSP PTAS for a MWPDV PTAS. (a) A minimum number of scans with the corresponding tour; (b) A tour with minimum tour length and the corresponding covering scan point set.

case the scan range is in the order of magnitude of the polygon's feature size. Both problems allow for a PTAS: Hochbaum and Maass [HM85] gave a PTAS for minimum cover, Arora [Aro98] and Mitchell [Mit99] presented polynomial-time approximation schemes for the TSP. Thus, the intuitional approach would be to use the PTAS for minimum cover to compute a set of scan points and compute a tour on these points with the second approximation scheme. However, we do not obtain a PTAS for the MWPDV by this combination. To see this, we consider two MWPDV (with unit rectangular visibility range) solutions for the example in Figure 4.1: In (a) a minimum number of scans with the corresponding tour is shown, in (b) a tour with minimum tour length and the corresponding scans that cover the polygon is depicted. Thus, using a PTAS to obtain a solution for one of the subproblems, we cannot combine this solution with a $(1 + \varepsilon)$-approximation for the other. In particular, because the relative weights of tour length and scan cost in the objective function influence on the optimal solution. So, there is no simultaneous PTAS for both problems. This naturally gives rise to the bicriteria problem stated in Section 4.1.1. With the approximation algorithms presented in Sections 4.3 and 4.4 we are able to give constant-factor approximations for the combined version (as well as for the bicriteria problem).

Moreover, when the scan range r is large in comparison to the minimum side length of the polygon, a, we may actually face the art gallery problem, see Figure 4.2. In case the scan range is sufficiently large, such that taking scans at the two points on the left allows for a perception of the niches (called "wells" in [AMP10]), locating scan positions at these points would suffice to cover all niches (the visibility polygons of the two points meet or even overlap in all niches). But, the locations of this pair of points must not coincide with vertices of the polygon, vertices of the polygon's arrangement of visibility polygons, they must not even be located on diagonals in the polygon. So, we do not have certain candidate points to choose from.

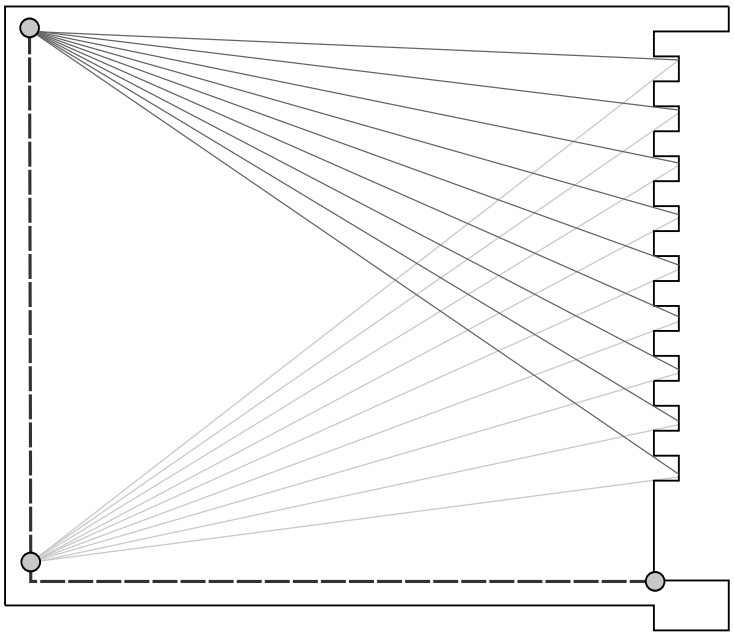


Figure 4.2: A minimum guard cover (points) with the corresponding tour length (dashed) in a polygon. The gray lines indicate lines of sight from the two left points.

As mentioned in Section 4.1.2 the Art Gallery Problem is NP-hard for several classes of polygons and variants on the placement of guards, see for example Lee and Lin [LL86]. For certain classes of polygons a minimum set of guards can be found in polynomial time. For example, Keil [Kei86] gave an $O(n^2)$ algorithm for covering a horizontally convex orthogonal polygon with star-shaped polygons. For the general problem, neither constant-factor approximation algorithms nor exact solution methods are known. Approximation algorithms are only known for restricted versions of the problem. As described in the preceding section, Efrat and Har-Peled [EHP06] gave an $O(\log OPT)$-approximation when the candidate guard points are limited to, for example, the vertices of the polygon.

In addition, for our problem we have to consider the objective function: We are only interested in a small guard cover in case the cost for scan points dominates the tour length. If, on the other hand, the tour length dominates the objective function, we may want to follow a tour alongside the niches (at the right side of Figure 4.2).

Consequently, for general polygons we will bound the ratio $\frac{r}{a}$. We will give a constant-factor approximation for a bounded value of $\frac{r}{a}$ in Section 4.5.

4.2 NP-Hardness

In this section we show that even the simplest variants of MWPDV milling are NP-hard.

Theorem 8. *MWPDV milling is NP-hard, even for rectilinear grid polygons, a uniform rectilinear visibility range and*

1. *small or no scan cost, that is, $c \ll 1$ or $c = 0$*
2. *small travel cost, that is, $c \gg 1$*
3. *no travel cost, that is, $t(T) = |\mathcal{S}(T)|$*

Proof. The first claim is a direct consequence of the NP-hardness of minimum cost milling, proven by Arkin et al. [AFM00]. The proof given by Arkin et al. uses the reduction from the problem `Hamiltonian circuit in planar bipartite graphs with maximum degree 3` to the problem `Hamiltonian circuit in grid graphs` (HGG), see Itai et al. [IPS82]. A grid graph is a node-induced finite subgraph of the infinite grid. For this reduction a parity-preserving embedding of the bipartite planar graph B into a rectangular grid graph G is used. The final considered grid graph G_9 is obtained by multiplying the scale of G by 9. Vertices in the constructed graph correspond to 9-clusters (the 9 nodes of a 2×2-square), edges are represented by *tentacles*, that is, unions of rectangular graphs with minimum dimension 2. So, starting with a graph B with n vertices, G_9 has $m = O(n)$ vertices. The tentacles can be covered by two Hamiltonian paths: a cross path (starting and ending in vertices corresponding to the 9-cluster of two different vertices of G) or a return path (starting and ending in vertices corresponding to the 9-cluster of one vertex of G). In case the edge is in the Hamilton circuit in B the cross path is used, otherwise the return path is used. The 9-clusters can only be covered by a Hamiltonian circuit if it is incident to exactly two cross paths. For the NP-hardness proof of minimum cost milling for an aligned unit square cutter, Arkin et al. defined the polygon to be the union of all placements of the cutter at the vertices of G_9. Then, if there is a tour of length m in G_9, there is a lawn mower (and also a milling) tour of length m. A lawn mower/milling tour of length m exists only when no point in the polygon is covered more than once. Consequently, the tour induces a partition of the polygon in non-overlapping rectangles of width 1 and integer length. Traversing a strip is equivalent to visiting the corresponding grid vertices of G_9. Consequently, a lawn mower/milling tour of length m induces a tour of length at most m in G_9.

When scanning infers small or no scan cost for the MWPDV, we ask for a shortest tour, such that (an arbitrary number of) scan points along this tour allows for coverage of the polygon. Consequently, we ask for the shortest milling tour in G_9. Hence, the NP-hardness is a direct consequence from the minimum cost milling reduction described above.

The second claim can be directly deduced from the NP-hardness of Hamiltonicity of Grid Graphs, proved by Itai et al. [IPS82] as described in the preceding paragraphs. We turn a given instance G of HGG with n vertices into an instance of MWPDV: We scale the grid graph G by a factor of two and substitute each grid point of G for a 2x2-square. Such, we obtain a set of n scan point that is contained in any optimal MWPDV solution. Visiting these scan points with a tour of length $2n$ is possible if and only if there exists a Hamilton circuit in G.

In the problem addressed with the third claim we aim at covering the polygon with scans only. Hence, the problem is obviously closely related to a minimum cover problem with (visibility L_∞-) disks. Though, for the MWPDV only the centers of the disks have to be located in the polygon's interior. The proof of this claim is based on a reduction of the NP-hard problem `Planar 3SAT`, a special case of `3SAT` in which the variable-clause incidence graph H is planar. The proceeding is along the lines of Baur and Fekete [BF01]. First, we construct a planar layout of H, for example by the method given by Rosenstiehl and Tarjan [RT86]. We will turn this embedding into a rectilinear grid polygon: We represent the variables, clauses and edges by polygonal pieces.

See Figure 4.3 for an example for the variable component. There are two ways to position a minimum number of scan points for the variable gadgets. One possibility—the black points in Figure 4.3 (a)—corresponds to a truth setting satisfying the clause for this variable. That is, a setting of "true" if v_i is in the clause, a setting of "false" if $\neg v_i$ appears in the clause. The other possibility—the circles in Figure 4.3(a)—corresponds to a truth setting not satisfying the clause. A connection of an edge corridor is shown in Figure 4.3(b): For the variable setting satisfying the clause the scan squares are pushed further into the edge corridor. See Figure 4.3(c) and (d) for the scan square positions at the edge corridor connection for the two truth settings. To guarantee the same number of points and circles, edge corridors may be added that do not end in another polygonal piece, but assure this parity (with circles at the edge corridor). The edge corridors are constructed analog to the variable components, in particular bendings are constructed accordingly.

A clause gadget is shown in Figure 4.4. The edge corridors of the three associated variables meet in the polygonal piece for the clause (depicted dark

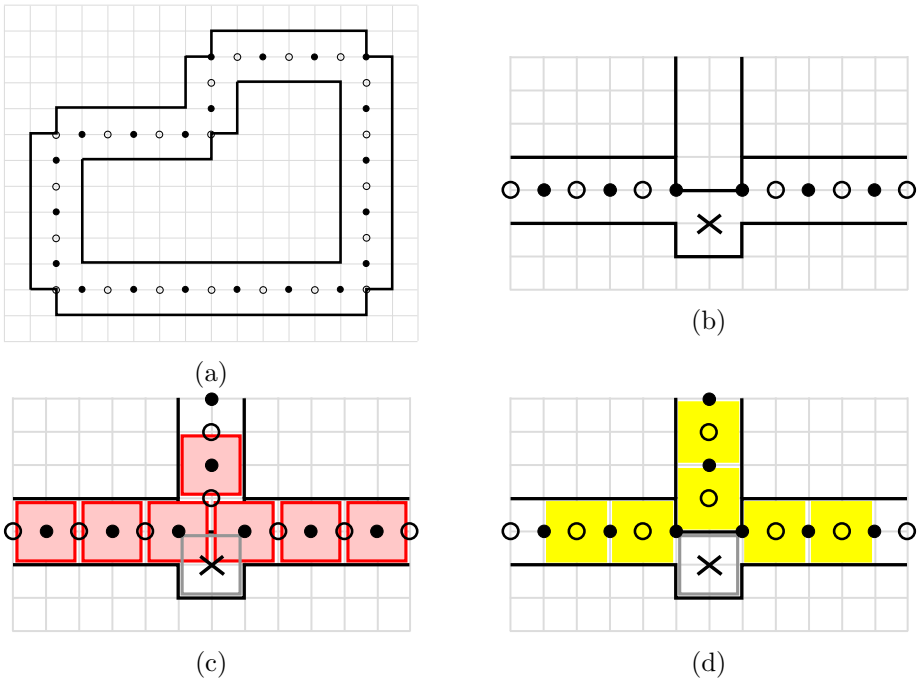


Figure 4.3: Top row: (a) The polygonal piece for a variable and (b) the connection of an edge corridor. Bottom row: a placement corresponding to a truth setting satisfying the clause (c), and a placement corresponding to a truth setting not satisfying the clause (d). The light gray scan is used in both cases.

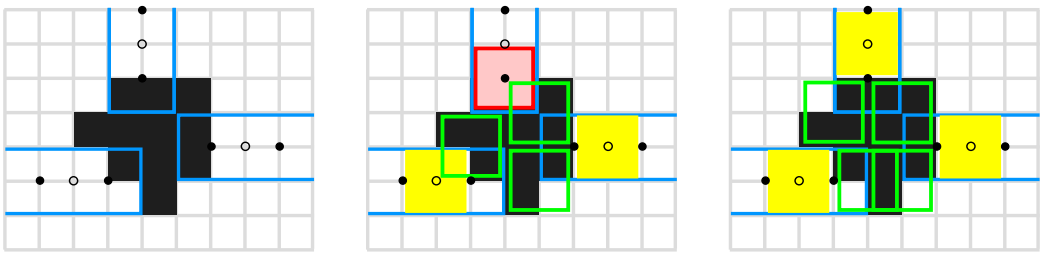


Figure 4.4: A clause component with the polygonal piece (dark gray) and the three according edge corridors (blue)(a). A placement corresponding to a truth setting satisfying the clause (b), and a placement corresponding to to a truth setting not satisfying the clause (c). The green squares are the scans necessary inside the clause.

gray in Figure 4.4). Three additional scans are sufficient to cover this polygonal piece if and only if the clause is satisfied, that is, if at least in one edge corridor the scan square is located at the black point. Otherwise, four scans are necessary.

With these components we can compute the parameter k, the number of square scans necessary to cover the resulting rectilinear grid polygon P. k is polynomial in the number of vertices of H and k is part of the input. All vertices of P have integer coordinates of small size. The number of vertices of P is polynomial in the number of vertices of H. So, we can conclude that the problem is NP-hard.

□

4.3 Approximation for Rectilinear MWPDV Milling for Rectangular Scan Range

The first approximation algorithm that we present is for the case of a uniform rectilinear visibility range and rectilinear grid polygons. The tour length is measured according to the L_1-distance. The approximation is for the case that no starting point is given. This algorithm already provides some general ideas for the other cases that we approximate.

The next lemma allows us to locate scan points at grid points only:

Lemma 1. *For a rectilinear grid polygon P there exists an optimum myopic watchman tour with discrete vision T^* such that all (unit L_∞-) scan points are located on grid points:*

$$\exists T^* \forall (x_s, y_s) \in \mathcal{S}(T^*) : x_s \in \mathbb{Z} \wedge y_s \in \mathbb{Z}. \tag{4.1}$$

Proof. Let T be an optimal tour, with (some) scan points not located on grid points. First, we shift the scans horizontally, such that the x-coordinates are integers: starting from the boundary, i.e., with distance 1 to the boundary if possible (this is not possible in case the width of the polygon is only a pixel), and away from non-reflex corners. The tour will not be longer and we do not cover less. In case we are able to reduce the number of scans we have a contradiction to T being optimal. The same is true if we shorten the tour by shifting horizontally. After shifting horizontally, we proceed analogously for the vertical direction. Hence, we have an optimal tour, with all scan points located on grid points. □

We approximate the MWPDV solution in two steps:

1. We construct a set of scan points that has not more than 2.5 times the number of scan points in a set of minimum cardinality.

2. We construct a tour that contains all scan points located in 1. and that is not longer than 2.5 times the length of an optimum milling tour.

We start with the second step. The basic idea for the tour originates from the 2.5-approximation for the milling problem from Arkin et al. [AFM00]. For the milling problem a solid shape (e.g., a circle or a square)—the "cutter"—has to cover every point in a given region. The problem asks for the shortest tour along which the cutter moves that allows for complete coverage. The cutter has to stay within the region. When we interpret our visibility range as a cutter, we do not face the constraint that the entire cutter must stay within the region—only the center point, the robot itself, must be located in the polygon at all times. In case of corridors of width 1 we extend the notion for the milling tour accordingly.

Let L_{OPT} denote the optimal milling tour length (as defined above). The tour is constructed from three parts, see Figure 4.5 for an example.

- *A "boundary" part*: We consider the inward offset region of all points in our polygon P that are feasible locations for the cutter. Let this region be $B(\subset P)$. Moving along the boundary δB of B a region in P is milled, we let this region be denoted by $P_{\delta B}$ (shaded in light blue in Figure 4.5). In case P features holes, δB is not connected and we denote the pieces by δB_i. In case of corridors of width 1 we connect the parts of δB by shortest paths. (In addition, for corridors of P of width 1 that do not connect two δB_is, we use shortest path on the grid such that every pixel of these corridors is adjacent to a part of this extended δB). The total length, $L_{\delta B}$, is a lower bound on L_{OPT}: $L_{\mathrm{OPT}} \geq L_{\delta B}$. This "boundary" part is shown in pink in Figure 4.5.

- *A "strip" part*: The strips are constructed to cover the still not watched interior, depicted in white in Figure 4.5, if it exists. Thus, we consider $P_{\mathrm{int}} := P \backslash P_{\delta B}$. We cover this region with a set of k horizontal strips str_i $(i = 1, \ldots, k)$. The y-coordinates of these strips differ by multiples of 2. The total length of these strips, $L_{str} = \sum_{i=1}^{k} L_{str_i}$, is then again a lower bound for L_{OPT}: $L_{\mathrm{OPT}} \geq L_{str}$. This "strip" part is shown in violet in Figure 4.5.

- *A "matching" part*: The "boundary" and "strip" parts described above allow for complete visibility coverage of P, but we are still left with the task to combine those tour components for a closed tour. This third and

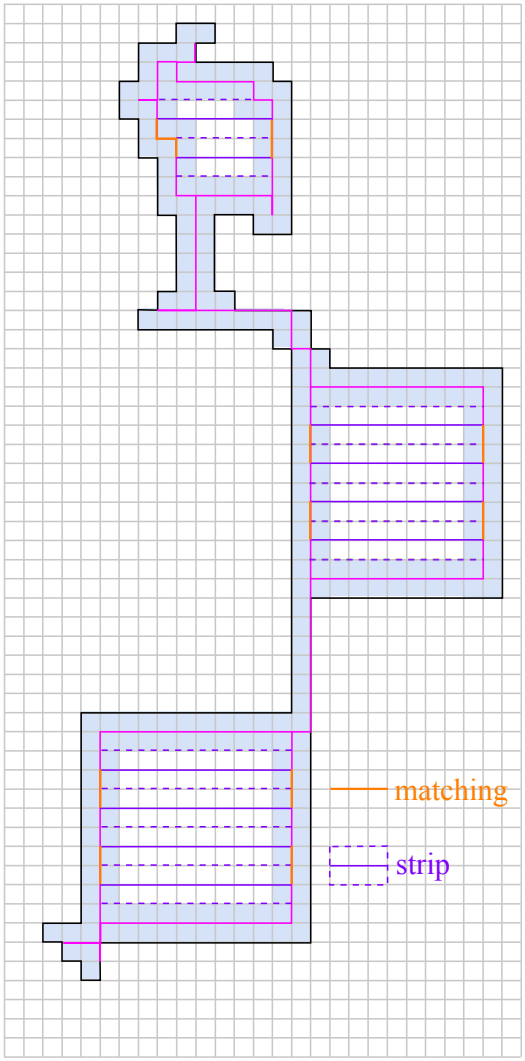


Figure 4.5: A rectilinear grid polygon P. The parts for the approximative milling tour are indicated: region $P_{\delta B_i}$ is shaded (light blue), δB_i is highlighted in pink.

last "matching" part serves as this connecting element. We consider the endpoints of the strips on δB_i. Every δB_i is partitioned in two disjoint portions by these endpoints. We denote these portions by $M_1(\delta B_i)$ and $M_2(\delta B_i)$. Using the shorter of these two ($M_*(\delta B_i)$) for every δB_i, we obtain for the combined length, L_M: $L_M \leq L_{str}/2 \leq L_{\text{OPT}}/2$. This "matching" part is shown in orange in Figure 4.5.

In the description of the "matching" part we justified its construction by the claim that it allows us to combine the other two components for a closed tour. To see this, we consider the graph G with vertices for endpoints of strip

lines and points where a strip line touches a δB_i. The vertices are connected by three different edge types: center lines of strips, the (parts of the) δB_is and the $M_*(\delta B_i)$s. In G every vertex has degree 4, hence we may construct an Eulerian tour and this yields a feasible solution.

Having constructed the tour, we proceed with the first step and construct a covering set of scan points, see Figure 4.6.

i. Consider the 2×2-squares that are fully contained in P and have two even coordinates. Let $S_{4,e}$ denote the centers of these "even quadruples". Remove all 2×2-squares that are centered in points from $S_{4,e}$, depicted in gray in Figure 4.6. Let $P_{\setminus\{(4,e)\}}$ denote the remaining rectilinear grid polygon.

ii. In $P_{\setminus\{(4,e)\}}$ greedily pick a maximum disjoint set $S_{4,o}$ of "odd quadruples": 2×2-squares that are fully contained in P and have at least one odd coordinate, shown in green in Figure 4.6. Let $P_{\setminus\{(4,e)\cup(4,o)\}}$ denote the remaining rectilinear grid polygon.

iii. In $P_{\setminus\{(4,e)\cup(4,o)\}}$ greedily pick a maximum disjoint set S_3 of "triple" scans: 2×2-squares that cover 3 pixels of $P_{\setminus\{(4,e)\cup(4,o)\}}$. These triple scans are shown in blue in Figure 4.6. Let $P_{\setminus\{(4,e)\cup(4,o)\cup(3)\}}$ denote the rectilinear grid polygon that remains after deleting the pixels corresponding to 2×2-squares centered in S_3.

iv. In $P_{\setminus\{(4,e)\cup(4,o)\cup(3)\}}$ no 3 pixels can be covered by a single scan (a 2×2-square). Consider the remaining pixels as vertices of a graph H. Two vertices in H are connected by an edge whenever the corresponding pixels may be covered by the same scan (whenever a 2×2-square can cover both pixels). Compute a maximum matching in H. This yields a minimum set of scans in $P_{\setminus\{(4,e)\cup(4,o)\cup(3)\}}$: The scans corresponding to the matching cover two pixels, those are "double" scans S_2, the remaining pixels are covered by "single" scans S_1. The double and single scans are depicted in red and yellow, respectively, in Figure 4.6.

Claim 6. *The total number of scans is at most 2.5 times the size of a minimum cardinality scan set.*

Proof. Let $s_{\min}$ denote the size of a minimum cardinality scan set. The 2×2-squares corresponding to $S_{4,e}$ and $S_{4,o}$ are fully contained in P. Thus, as no scan of the optimum can cover more than 4 pixels, we have $|S_{4e} \cup S_{4o}| \leq s_{\min}$.

So, we still need to bound $|S_3 \cup S_2 \cup S_1|$. In $P_{\setminus\{(4,e)\cup(4,o)\}}$ a single scan (a single 2×2-square) cannot cover more than 3 pixels. In $P_{\setminus\{(4,e)\cup(4,o)\cup(3)\}}$ we

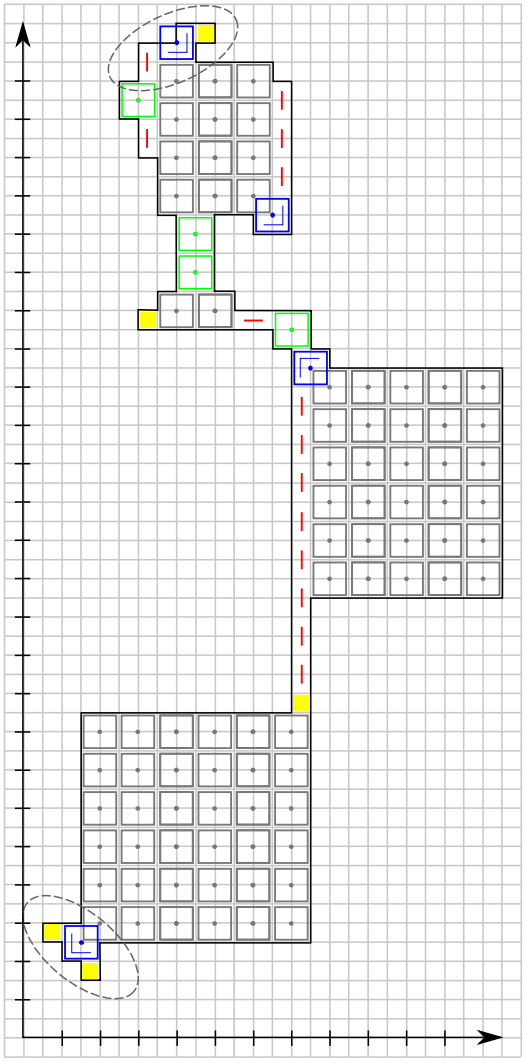


Figure 4.6: An example for our approximation method (polygon P from Figure 4.5): The set of "even quadruple" scans is shown in grey; the "odd quadruple" scans are green. A possible (greedy!) set of "triple" scan is shown in blue, leaving the maximum matching (and the corresponding "double scans") shown in red. The leftover single pixels are yellow. The ellipses indicate a part that is covered by three scans instead of two. The triple scan with adjacent single and double scans could be covered by two triple scans. The triple scan with adjacent single scans could be covered by a triple and a double scan.

compute an optimal solution. Thus, the only way we could have improved our cover for $P_{\setminus\{(4,e)\cup(4,o)\}}$ is by selecting a different set of triple scans (i.e., triple scans at different positions). $P_{\setminus\{(4,e)\cup(4,o)\}}$ does not contain any 2×2-squares. Hence, a simple case analysis shows that only two possible improvements are possible:

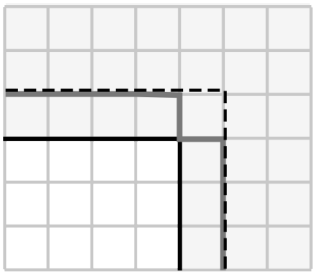


Figure 4.7: Modifying the boundary part at a reflex vertex, the gray path is used instead of δB_i.

- replacing a triple, a double and a single by two triples (as shown in the upper ellipse of Figure 4.6), or
- replacing a triple and two singles by a triple and a double (as shown in the lower ellipse of Figure 4.6).

Let $s^4_{\min}$ be the minimum number of scan points required for covering $P_{\setminus\{(4,e)\cup(4,o)\}}$. The case analysis yields $|S_3 \cup S_2 \cup S_1| \leq \frac{3}{2}s^4_{\min} \leq \frac{3}{2}s_{\min}$.

Thus, altogether we have: $|S_{4e} \cup S_{4o}| + |S_3 \cup S_2 \cup S_1| \leq s_{\min} + \frac{3}{2}s_{\min} = \frac{5}{2}s_{\min}$. That is, in total we construct a set of scan points with no more than 2.5 times the number of scans in a minimum cardinality scan set.

□

Claim 7. *All scan points constructed in step 1 lie on a 2.5-approximative milling tour, as constructed in step 2.*

Proof. We choose for the center lines of strips to be located on even y-coordinates. Thus, we visit all scan points in S_{4e} by tracing the strips. In addition, all pixels in $P_{\setminus\{(4,e)\}}$ are either boundary pixels or are adjacent to boundary pixels in P. Consequently, all these pixels may be visited when tracing the "boundary" part of the tour—one minor adjustment of this tour may be necessary for triple scans. To visit the center of a triple scan located on a reflex vertex we need to reroute the boundary part as shown in Figure 4.7, this does not influence on the tour length. □

This concludes the proof for the approximation factor of our algorithm. We summarize:

Theorem 9. *A rectilinear grid polygon P allows for a MWPDV milling with rectangular vision solution that uses no more than 2.5 times the number of scans in a minimum cardinality scan set and has a tour length that does not exceed 2.5 times the length of an optimum milling tour.*

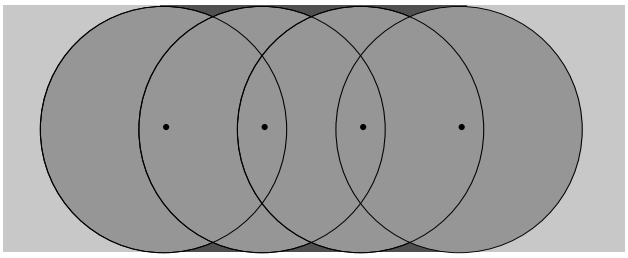


Figure 4.8: A finite number of scan points along the center line of a corridor of width $2r$ does not suffice to cover the corridor with a circular scan range of radius r.

4.4 Approximation for Rectilinear MWPDV Milling for Circular Scan Range

The approximation algorithm we present in this section is for the case of a uniform circular range of visibility and rectilinear grid polygons. The tour is again measured according to the L_1-distance. The approximation is for the case that no starting point is given or for the case of a given starting point located on a grid point (in the rectilinear grid polygon or on its boundary).

Before we give the details of our approximation, note an additional difficulty of a circular scan range. Using a circular scan range of radius r with continuous vision allows to cover a corridor of width $2r$ by moving along the corridor's center line. In contrast, using only discrete scan points, we face boundary effects: A finite number of discrete scan points located on the corridor's center line do not allow for coverage of the total width of $2r$, see Figure 4.8. Consequently, we have to treat this effect.

Again, we approximate the MWPDV solution in two steps.

1. In the first step we construct a set of scan points that covers P.

2. In the second step we construct a tour that visits all scan points located in step 1.

We start with the construction of a covering set of scan points. We overlay the rectilinear grid polygon with a point grid as in Figure 4.9, left. That is, we apply a diagonal point grid with L_2-distance of $\sqrt{2}$ in-between points, and use all points of this grid that coincide with a grid point in P as scan points (drawn in dark gray in Figure 4.9, left).

For the second step, the tour T, our strategy starts at a boundary grid point. We then construct our tour counterclockwise along the boundary (taking a scan at every point of the overlayed diagonal points grid located on the boundary). We still have to expand our tour, because with this construction

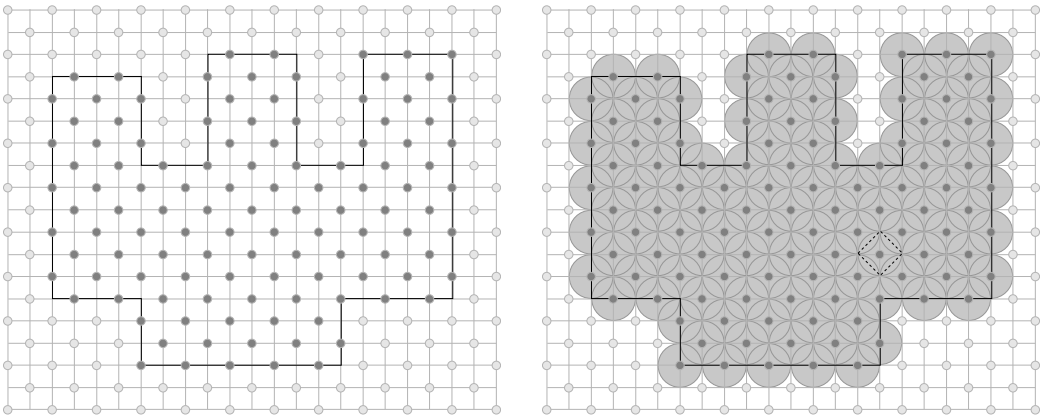


Figure 4.9: Left: Point grid (light gray) with grid points within a rectilinear grid polygon (black) in dark gray. Right: Circular visibility ranges of the grid points covering the plane, one square of side length $\sqrt{2}$ is indicated by a dashed line.

we would be left with a tour covering an area around the boundary. Thus, we include movement in-between interior scan points.

We use horizontal strips located on grid lines (that is, the y-coordinates of these strips differ by 1) and distance 1 to the boundary, shown in yellow in Figure 4.10 for an example. In order to combine these grid lines with the boundary path for a tour, we link strips to the left boundary. Two of these will in general be linked on the right-hand side. Those connections are depicted in solid blue in Figure 4.10. In case there is an odd number of strips between the upper and lower boundary of P, the scan points located on the bottommost strip are visited by using a path of (L_1-)length 2, from the boundary or strips with another y-coordinate of the leftmost point, see Figure 4.10, down right. The strips that get linked are always determined by the leftmost boundary. In case there are other parts of P that have a left boundary (that is vertical edges with polygon to the right and the exterior to the left) whose cardinality of strip lines differ by an odd number, scan points from the topmost strip are linked by two vertical steps of length 1 to the upper boundary, see Figure 4.10, top right. These (L_1-)length 2 connections are drawn in dash-dotted blue in Figure 4.10. So, we yield a closed tour: We always link two strips, hence, we always end up at the left boundary, the rest is a tour along the boundary with small loops of length 2. Moreover, as it was constructed like this, the tour visits all scan points located in step 1.

We still need to show that P is covered and have to consider the approximation factor of our strategy. Let T be the tour determined by our strategy, and T^* be an optimal tour.

Lemma 2. *The scan points the strategy positions (in Step 1) cover P.*

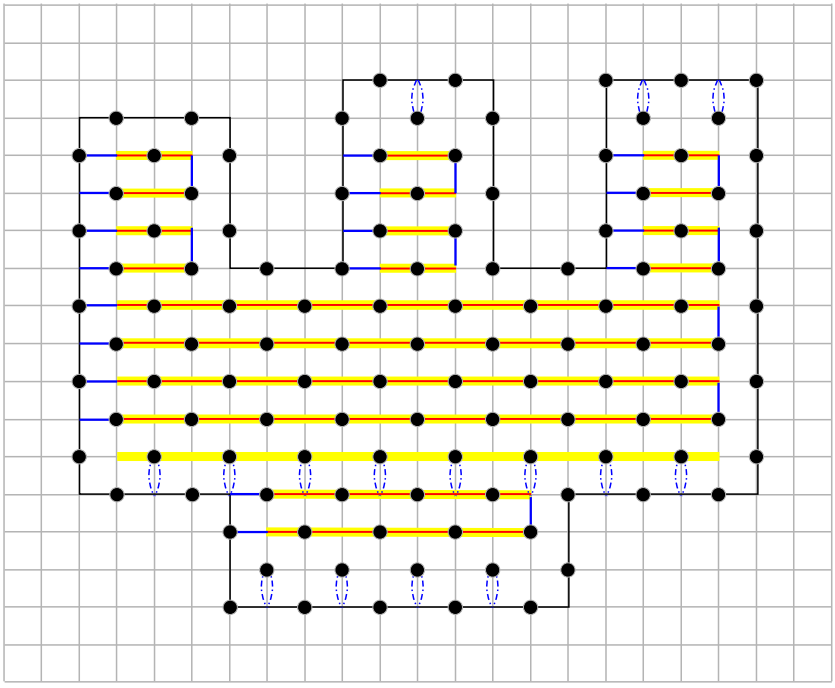


Figure 4.10: A rectilinear grid polygon P with the tour given by our strategy. Scan points are displayed in black. The horizontal strips of total length L_{strips} are indicated in yellow, the tour is in blue for the links to the boundary and in-between strips (solid) as well as for connections of points (dash-dotted), parts located on the strips are indicated in red. (In-between those parts the tour runs on the boundary.)

Proof. The scan range is a L_2-circle of range r. The scan points are located in a diagonal point grid with L_2-distance of $\sqrt{2}$ in-between points. Each circle covers at least a square of side length $\sqrt{2}$, with the vertices located on grid points that do not belong to the overlayed grid, see Figure 4.9 right. □

Lemma 3. $L(T) \leq 4 \cdot L(T^*) + 8$

Proof. Our tour consists of the tour along the boundary of P, the strips and connections.

Let $L_{\delta P}$ denote the length of P's boundary, i.e., its perimeter. Let $L_{\delta B}$ be defined as in Section 4.3. Moreover, let the length of the horizontal strips be $L_{str(y-dist1)}$.

We show that the length of our tour T does not exceed the combined length of these three components. The parts of T running along the boundary are covered by the δP. All parts of T that are located on the strips are covered by the part of length $L_{str(y-dist1)}$. The right-hand side links of two adjacent strips lie on δB. The left-hand side connections can be turned by 90° (only for charging) and lie on δ_B as well. The scan point connecting paths of length

2 can be swung open and lie on the strip of the relative (not used) strip or one part (length 1) on δ_B for interior strip points or endpoints of strips, respectively. Consequently, the sum of $L_{\delta P}$, $L_{\delta B}$ and $L_{str(y-dist1)}$ is an upper bound for our tour length, $L(T)$: $L(T) \leq L_{\delta P} + L_{\delta B} + L_{str(y-dist1)}$.

For an orthogonal polygon of n vertices, r of which are reflex, we have $n = 2r + 4$ (see O'Rourke [O'R87], Lemma 2.12). Hence: $L_{\delta P} = L_{\delta B} + 8$. Furthermore, we have $L_{str(y-dist1)} \leq 2 \cdot L_{str} \leq 2 \cdot L(T^*)$, with L_{str} as defined as in Section 4.3. Finally, the milling argument still yields $L_{\delta B} \leq L(T^*)$.

When we combine these bounds, we obtain:

$$\begin{aligned} L(T) &\leq L_{\delta P} + L_{\delta B} + L_{str(y-dist1)} \\ &\leq 2 \cdot L_{\delta B} + 8 + L_{str(y-dist1)} \\ &\leq 4 \cdot L(T^*) + 8. \end{aligned}$$

□

Lemma 4. $|\mathcal{S}(T)| \leq 4 \cdot |\mathcal{S}(T^*)|$

Proof. Let $N(P)$ be the number of pixels of a rectilinear grid polygon P. To be precise we write $\mathcal{S}_P(T)$ (instead of $\mathcal{S}(T)$) to indicate the polygon we refer to for the scan points. We then define $R(N(P))$ as follows:

$$R(N(P)) = \max\{\frac{|\mathcal{S}_P(T)|}{|\mathcal{S}_P(T^*)|} | P \text{ has } N(P) \text{ pixels}\}$$

Kershner [Ker39](see Tóth [Tót49]) showed for $D(r) = \frac{\pi r^2 |\mathcal{S}_P(T^*)|}{N(P)}$:

$$D(r) \geq \frac{2\sqrt{3}\pi}{9}$$

So, for $r = 1$, we obtain:

$$|\mathcal{S}_P(T^*)| \geq \frac{2\sqrt{3}}{9} \cdot N(P)$$

With our approximation algorithm we place scan points on the diagonal grid, so there is an upper bound of $2 + (N(P) - 1) = N(P) + 1$ for the scan points we use. Consequently, an upper bound on $R(N(P))$ is given by

$$\frac{N(P) + 1}{\frac{2\sqrt{3}}{9} \cdot N(P)}.$$

We define $F(n) := \frac{n+1}{\frac{2\sqrt{3}}{9} \cdot n}$. $F(n)$ is monotonically decreasing in n. Moreover:

$$F(n) = 4 \Leftrightarrow n = \frac{3\sqrt{3}}{8 - 3\sqrt{3}} \approx 1.85322$$

Thus, we yield

$$R(N(P)) \leq F(N(P)) \leq 4 \text{ for } N(P) \geq 2. \tag{4.2}$$

We may conclude

$$\frac{|\mathcal{S}_P(T)|}{|\mathcal{S}_P(T^*)|} \leq 4 \text{ for } N(P) \geq 2. \tag{4.3}$$

For $N(P) = 1$ the optimum needs at least one scan, we place at most 2 scans. □

Theorem 10. *There exists a 4-approximation for the MWPDV milling for a circular visibility range with $r = 1$ for a rectilinear grid polygon P.*

Proof. With Lemmas 3 and 4 we have:

$$\begin{aligned} t(T) &= c \cdot |\mathcal{S}(T)| + L(T) \\ &\leq c \cdot 4 \cdot |\mathcal{S}(T^*)| + 4 \cdot L(T^*) + 8 \\ &= 4 \cdot (c \cdot |\mathcal{S}(T^*)| + L(T^*)) + 8 \\ &= 4 \cdot t(T^*) + 8. \end{aligned}$$

□

4.5 Approximation for General MWPDV Milling for Circular Scan Range

In this section we present an approximation algorithm for the case of a circular visibility range r and general polygons. Again, we consider MWPDV milling. As discussed in Section 4.1.3, even the problem of minimum guard coverage has no known constant-factor approximation; therefore, we consider a bounded ratio r/a between visibility range and feature size, i.e., minimum side length. The approximation is for the case that no starting point is given.

As for the cases presented in Sections 4.3 and 4.4, we construct our approximation for MWPDV milling in two steps:

1. We construct a set of scan points $\mathcal{S}(T)$ that is within a constant factor of the number of scan points in a set of minimum cardinality.

2. We construct a tour T that contains all scan points located in 1. and that is within a constant factor of the length of an optimum milling tour.

Again, we start with the second step, and construct a tour. This tour will then form the basis for the placement of scan points. Just as in the case with a rectilinear visibility range and rectilinear grid polygons of Section 4.3, our tour is constructed from three parts.

- *A "boundary" part*: In Section 4.3 we described tracing δB, causing a tour length of $L_{\delta B}$. Here, we use two "boundary tours"—within distance of (at most) $\frac{1}{2}r$ and (at most) $\frac{3}{2}r$ to the boundary. Let these tours be denoted by $T_{1/2r}$ and $T_{3/2r}$, with length $L_{T_{1/2r}}$ and $L_{T_{3/2r}}$, respectively. Then we have:

$$L_{T_{1/2r}} + L_{T_{3/2r}} = 2 \cdot L_{\delta B} \leq 2 \cdot L(T^*) \tag{4.4}$$

  (The length of the three tours differs at the vertices: Drawing a line perpendicular there from $T_{3/2r}$ to $T_{1/2r}$ the Intercept Theorem shows that the distance to the diagonal through the vertices of all tours on $T_{1/2r}$ is twice as much as on the boundary tour with distance r to the boundary.)
  The two "boundary" tours allow us to cover a corridor of width $2r$ with a bounded number of scans, while (4.4) enables us to bound the tour length in terms of the optimal length.

- *A "strip" part*: Again we construct strips for the interior, $P_{\text{int}} := P \backslash P_{\delta B}$. If P_{int} is nonempty it can be covered by a set of k_1 horizontal strips Str_i^1 $(i = 1, \ldots, k_1)$ (with continuous vision those would really allow for coverage). The y-coordinates of two of these strips differ by multiples of $2r$. We consider another set of k_2 strips, Str_i^2 $(i = 1, \ldots, k_2)$, shifted by r. Then, let $L_{str}^j = \sum_{i=1}^{k_j} L_{Str_i^j}$. Analog to the argument for L_∞ scan range, we have $L_{str}^1 + L_{str}^2 \leq 2 \cdot L(T^*)$.

- *A "matching" part*: Again, this third and last "matching" part serves as the element that combines the "boundary" and "strip" parts for a closed tour. Here, we add two more set of sections:

  - The center lines of the strips have a distance of r to the boundary, thus they do not yet touch $T_{1/2r}$. Consequently, we add $1/2r$ to each center line (on each end). For that purpose, we consider the matchings as defined in Section 4.3. (Recall the definition: Consider the endpoints of the strips on δB_i. Every δB_i is partitioned in two disjoint portions by these endpoints. We denote these portions by $M_1(\delta B_i)$ and $M_2(\delta B_i)$. Using the shorter of these two $(M_*(\delta B_i))$ for every δB_i, we obtain for the combined length, L_M:

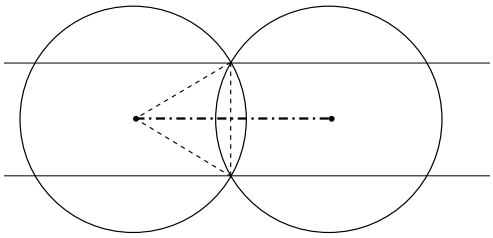


Figure 4.11: Two scan points with distance $\sqrt{3} \cdot r$ to each other: The dashed lines indicate a length of r, so, the dash-dotted line has a length of $\sqrt{3} \cdot r$, the distance of two scans.

$L_M \leq L_{str}/2 \leq L_{OPT}/2$.) Because two strips are at least a distance of r apart, the connection to $T_{1/2r}$, $C_{(1/2r)}$, costs less than $1/2 \cdot L_M \leq 1/2 \cdot L_{str}/2 \leq L(T^*)/4$.

– Moreover, we consider the above matchings defined on $T_{1/2r}$ and use the shorter sections of the disjoint parts, $(M_{*(1/2r)}(\delta B_i))$, for every δB_i. The Intercept Theorem in combination with the analogously defined sections on $T_{3/2r}$ enables us to give an upper bound of $L_{M_{(1/2r)}} \leq L_{str} \leq L(T^*)$.

Starting on some point on $T_{1/2r}$, tracing the strips, and the inner "boundary" $T_{3/2r}$ at once when passing it, yields a closed tour. We summarize:

Lemma 5. $L(T) \leq 21/4 \cdot L(T^*)$

Proof.

$$\begin{aligned} L(T) &= L_{T_{1/2r}} + L_{T_{3/2r}} + L^1_{str} + L^2_{str} + L_{C_{(1/2r)}} + L_{M_{(1/2r)}} \\ &\leq 2 \cdot L(T^*) + 2 \cdot L(T^*) + L(T^*)/4 + L(T^*) \\ &= 21/4 \cdot L(T^*) \end{aligned}$$

□

Having constructed the tour, we only have to take care of Step 1, that is, we have to construct a covering set of scan points (that are all located on T as constructed above).

- For the "boundary" part we place scans with the center points located on $T_{1/2r}$ and $T_{3/2r}$ in distance $\sqrt{3} \cdot r$ (see Figure 4.11) if possible. However, at corners we need to place scans, so the minimum width we are able to cover with the two scans (on both tours) is a.
- For the "strip" part the distance of scans is also $\sqrt{3} \cdot r$ on both strip sets, exactly the distance enabling us to cover a width of r, see Figure 4.11.

It remains to consider the costs for the scans. Taking scans within a distance of $\sqrt{3}\cdot r$, we may need the length divided by this value, plus one scan. We only charge the first part to the strips. The (possible) additional scans are charged to the "boundary" part, as we have no minimum length of the strips. So, adding a constant cost would not allow us to stay within a constant factor of the number of scan points in a set of minimum cardinality. The optimum cannot cover more than πr^2 with one scan. Let $L_{str} = max(L^1_{str}, L^2_{str})$:

$$\begin{aligned} |\mathcal{S}(T^*)| &\geq \frac{L_{str}}{\pi r/2} \\ |\mathcal{S}(T)| &\leq \frac{2L_{str}}{\sqrt{3}\cdot r} \\ \Rightarrow \quad \frac{|\mathcal{S}(T)|}{|\mathcal{S}(T^*)|} &\leq \frac{2L_{str}}{\sqrt{3}\cdot r} \cdot \frac{\pi r/2}{L_{str}} = \frac{\pi}{\sqrt{3}} \end{aligned} \tag{4.5}$$

Finally, we consider the "boundary". We assume $L_{\delta B} \geq 1$. Hence:

$$|\mathcal{S}(T^*)| \geq \frac{L_{\delta B}}{\pi r/2}.$$

We may need to scan within a distance of a—on two strips—, need additional scans and have to charge the scans from the "strip" part, hence, this yields:

$$|\mathcal{S}(T)| \leq \frac{L_{\delta B}}{a/2} + 1 + \frac{L_{\delta B}}{r}.$$

Altogether ("boundary" and "strips"), we have for $r \geq a$:

Lemma 6. $|\mathcal{S}(T)| \leq |\mathcal{S}(T^*)| \cdot (\frac{\pi r}{a} + \frac{\pi r}{2} + \frac{\pi}{2})$

So, when we combine Lemmas 5 and 6, we obtain our approximation factor.

Theorem 11. *A polygon P allows a MWPDV solution that contains at most a cost of* $\max(\frac{21}{4}, \frac{\pi r}{a} + \frac{\pi r}{2} + \frac{\pi}{2})$ *times the cost of an optimum MWPDV solution (for $r \geq a$).*

Note that Theorem 11 covers the case from Section 4.4; however, instead of the factor 4 originated by the adaption to this special case, this yields a factor of $2 \cdot \pi$.

4.6 Conclusion

We studied the problem of guarding a polygon P with a mobile watchman with limited and discrete vision that has to stay within the polygon at all times.

We showed that even the simplest variants of MWPDV milling are NP-hard. Moreover, we presented approximation algorithms for different variants of the problem. All of these algorithms also apply for the bicriteria version, that is, considering tour and set of scan points separately. We gave a 2.5-approximation method for a watchman with uniform rectilinear visibility range in rectilinear grid polygons. For the surveillance of rectilinear grid polygons with a watchman with uniform circular visibility range we presented a 4-approximation. Finally, for a watchman with circular visibility range r monitoring general polygons we presented an approximation algorithm with factor of $\max(\frac{21}{4}, \frac{\pi r}{a} + \frac{\pi r}{2} + \frac{\pi}{2})$, a constant factor for a bounded ratio of r/a. This approximation is based on ideas of the simple 2.5-approximation. The basic concept we used for the approximation algorithms is a tour construction of the "boundary area" of the polygon, and a construction of an internal tour, that are then combined for a connected tour. Polygon covering scan points are located such that they can be visited by the constructed tours.

Note that for the lawn mowing variant of the MWPDV with combined cost (that is, not for the bicriteria version) Mitchell [FMS09] gave a PTAS based on guillotine subdivisions. This PTAS makes use of a fixed ratio between scan cost and travel cost.

As the factor for general polygons depends on r/a there is still room for improvement. In particular, it would be desirable to achieve, if not an approximation factor independent of r/a, a factor that depends on $\log(r/a)$. In general, the location of scan points includes a geometric hitting set problem. However, the objects that need to be covered are for the general case not r-admissible (for which Mustafa and Ray [MR09] gave a PTAS), not "fat" as defined by Chan [Cha03], do not fulfill the locality condition (another geometric setting for which Mustafa and Ray [MR09] presented a PTAS) and the VC-dimension does not allow for ε-nets of size $O(1/\varepsilon)$ (see Haussler and Welzl [HW86]) for which a constant-factor approximation for the hitting set problem presented by Brönnimann and Goodrich [BG94] exists. Hence, gaining independence of the factor of r/a is not likely.

CHAPTER 5

Exploration and Triangulation with a Swarm of Robots

This chapter deals with the problem of surveying a polygon with a swarm of mobile agents with limited communication range. The agents have to explore the yet uncovered polygon and decide on locations that allow for a triangulated network, with edge lengths restriction, covering the entire region. We seek for a minimal number of agents necessary for this task. Or, if only a limited of agents is available, we aim at maximizing the covered subpolygon.

5.1 Problem Description

Exploring an environment with a single robot, for example, by looking for a tour for the robot along which every point in the environment becomes visible at least once, is a problem we described in the last chapter. A solution to this watchman route problem allows for visibility coverage of all points in the environment, but a single point may be perceived at only one point in time. So, a possible goal is to ensure guarding at all times—like stated in the art gallery problem for mostly stationary guards, another classical scenario discussed in Chapter 4. These guards permanently monitor the region, but interaction and communication between the single guards is not taken into account. Moreover, the art gallery problem is not concerned with the process of positioning the guards at the chosen locations.

The capabilities of autonomous robots have improved in recent years. Moreover, the price of commercially available platforms (as the iRobot Room-

bas we use, depicted in Figure 5.1) decreased to a point that makes it interesting to consider exploration and guarding with a robot swarm. Though, these agents, the single robots, have only limited capabilities.

If we think of a static sensor network, used, for example, to track people or events, that needs to react to different scenarios by adding further mobile sensors, we want to keep the network properties. Thus, if we use sensor nodes attached to mobile robots, as shown in Figure 5.1, we aim at exploring a yet uncovered environment. But the coverage of already explored area should be permanent. Moreover, sensor nodes typically have a limited communication range. Consequently, if we want to assure a connected network, we cannot just focus on finding positions that cover the entire region (with a limited range of perception in contrast to most art gallery results), but the location process must maintain communication edges.

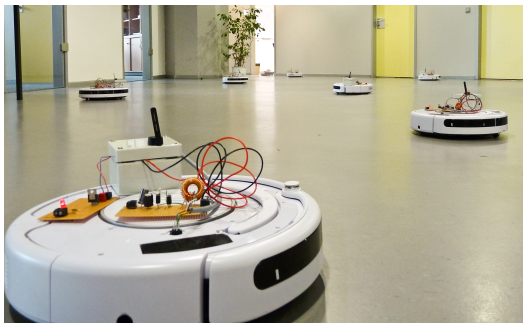


Figure 5.1: A robot swarm consisting of iRobot Roombas.

Thus, as in other classical exploration or guarding problems, we are given a known or unknown polygonal region P. The polygon needs to be fully monitored by a swarm of robots. For this monitoring task we want to use a well connected network that can perform the described location services. In particular, we ask for a triangulated network. Hence, agents with only limited input information, and without global coordinates, have to explore the environment and find positions that allow for a triangulated network that covers the entire area. Having only local information, the computation is carried out decentralized and by local cooperation.

If the task is to fully explore a region, a natural objective is the minimization of the number of robots. On the other hand, if we are equipped with only a limited number, ℓ, of robots, we aim at maximizing the subregion covered by the triangulation.

5.1.1 Problem Statement

We are given a polygon, P, that may be a polygon with holes. Moreover, we are given a starting point, s, on the boundary of the yet unexplored (connected) subpolygon $P_u \subseteq P$. For ease of description we consider only this subpolygon as our polygon and denote it by P.

Every robot in the swarm has a (circular) communication range r. Within this range, perception of and communication with other robots is possible. For the ease of description we assume that r is equal to 1 (and scale the polygon accordingly). We use the terms *robots* and *relays* synonymously.

The *Minimum Relay Triangulation Problem* (MRTP) asks for a triangulation that covers P with edge length at most 1. The objective is to minimize the number of relays. The triangulation must not contain edges crossing the boundary of P, reflecting the impossibility of communicating through walls. Thus, the triangulation contains all vertices of P, plus intermediate points. The latter are needed as edges in the triangulation must not have a length exceeding $r = 1$. The formal statement for the MRTP is: Given a polygon P and a starting point $s \in P$, determine a minimum-cardinality set R ($s \in R$) of relays in P such that there exists a triangulation of R that fully covers P and with edge lengths at most 1.

Given a polygon P and ℓ relays, the *Maximum Area Triangulation Problem* (MATP) asks for a connected triangulation of a subarea within P that maximizes the covered area. Hence, given a polygon P, a starting point $s \in P$ and ℓ relays, the MATP asks for a set R ($s \in R$) of $|R| = \ell$ relays in P, such that there exists a triangulation of R with edge lengths at most 1 that maximizes the covered area.

For the online versions (OMRTP and OMATP), the polygon P is unknown. Each relay moves through the polygon and decides on a new location while still within range of other relays. Once it has decided for a position and stopped, it will not move anymore and becomes part of the static triangulation. Such it allows for other relays to extend the explored area and the triangulation. This restriction is motivated by our application: in particular, if the area that we explore is huge, we wish to be able to begin location services in the triangulation that is already fixed. That is, even if the polygon is not yet fully explored—which may take a long time in case of large areas—the extended network can be used. More precisely, also for the OMRTP we are given a polygon P and a starting point $s \in P$. We aim at computing a minimum-cardinality set R ($s \in R$) of relays in P such that there exists a triangulation of R that fully covers P and with edge lengths at most 1. The relays move into the polygon, starting from s. A relay extending the yet established subset $R' \subset R$ must stay within a distance of 1 to at least

one relay $r \in R'$. Once it fixed its position it will not move again. The OMATP is defined analogously. No non-triangle edges are allowed in the final construction. For the OMRTP we let R_{OPT} denote the number of relays used by the optimum, for the OMATP A_{OPT} denotes the area covered by the optimum—both for the optimal triangulation T_{OPT}.

Organization of this Chapter. The remainder of the current section presents related work. In Section 5.2 we show that both the MRTP and the MATP are NP-hard. In Section 5.3 we follow the question which competitive ratio a strategy for the OMRTP can achieve. We show a lower bound of 6/5 for the competitive ratio of any online strategy. In addition, we present an algorithm that achieves a competitive ratio of 3. Moreover, we consider online strategies for the OMATP (Section 5.4). It turns out that no strategy can achieve a constant competitive ratio.

The results in this chapter are based on work together with Sándor P. Fekete, Tom Kamphans, Alexander Kröller, and Joseph S. B. Mitchell [FKK+11].

5.1.2 Related Work

As described in the beginning of this section, the MRTP and the MATP are both related to online exploration problems (in particular, with watchmen with limited vision) and to the placement of stationary guards, the art gallery problem. We discussed these problems in the preceding chapter, see Section 4.1.2 for a detailed overview on related work.

We aim at obtaining a triangulated network. In contrast to classical *triangulation* problems that ask for a triangulation of all vertices of a polygon (possibly allowing some additional *Steiner points*), we allow for triangles of a certain type (all edges have length ≤ 1) only. Triangulations with shape constraints for the triangles and the use of Steiner points are considered for *Mesh generation*. Though, the constraints differ from the ones we impose. For both MRTP and MATP we intend to place relays to obtain a connected network. In contrast the *relay placement* problem asks for locating relays (with limited communication range) such that a given set of sensors is connected via relay paths.

If we focus on distributing the robots of the swarm over the entire environment, we face a *dispersion* problem.

Finally, for the MATP we are equipped with a certain number of relays and want to cover as much area as possible. That is, we are given a certain budget and need to optimize the area collected. This is related to other optimization problems with a hard limit on the cost, especially the *orienteering* problem that asks for a limited length tour among a subset of points with

maximal reward. Because we are given a starting point, in particular the rooted orienteering problem with a given start (or root) node is of interest.

Another problem in this context of problems with a given budget is the search for a maximum area (or perimeter) convex k-gon using k out of n given points. Related tasks refer to enclosing a subset of a given set of reward-labeled objects with a closed curve, such that the enclosure's value (depending on enclosed points rewards and the cost for the curve) is maximized. These problems are called *geometric knapsack* problems.

Triangulation. Classical triangulation problems ask for a triangulation that includes all vertices of a polygon, but without any restriction on the edge length, see O'Rourke [O'R87] and de Berg et al. [dBCvKO08] for an overview. That is, in contrast to the limited edge length (given by the relay's communication range) we face, the triangulation may contain any edge that is fully contained in the given polygon. For simple polygons, Chazelle [Cha91] presented a linear time triangulation algorithm. Variants of the problem ask, for example, for a triangulation that minimizes the sum of edge lengths. Mulzer and Rote [MR08] proved the decision version of this problem to be NP-hard.

Meshing. Meshing is an essential preprocessing step for the finite element method. This method is used for approximating continuous problems by subdividing the given domain into a mesh of elements and approximating the given function by a piecewise polynomial on these elements, see Eppstein [Epp01] for an overview on meshing. Meshing is the process of partitioning the given domain into simple cells, the elements, of a certain kind, for example, triangles or quadrilaterals. The focus is on the approximation property. Thus, the mesh must contain the entire region. But, there may be parts of the domain with a more complicated solution than others, requiring a finer mesh in this regions in order to guarantee an acceptable approximation. Moreover, the number of elements, that is for us the number of triangles, should not be too large. The number of elements influences the solution time for the approximation step. Finally, the triangles should fulfill certain shape requirements. In general, small and obtuse angles should be avoided. Allowing for triangles with such angles would reduce the accuracy of the numerical solution. Bern and Eppstein [BE92] gave a survey on mesh generation. They discussed triangulation with and without Steiner points and with and without optimality criteria. For non-Steiner triangulations the focus is on the triangles' shape. So, quality measures rely on their angles, edge lengths, heights or areas. The criteria is then the sum, maximum or minimum of the measure over all triangles. For triangulations with Steiner points that either limit the number of Steiner points and then ask for an optimal triangulation, or that minimize the number of Steiner points for a

given criterion, approximation algorithms are presented. Criteria of interest include the maximization of the minimum angle, avoidance of large angles, the maximization of the minimum height and the minimization of the total edge length.

Algorithms surveyed by Bern and Eppstein include a technique by Chew [Che89] that guarantees all angles in the triangulation to be between 30 and 120 degrees and the edge lengths to be between h and $2h$ (with h being a parameter that can be chosen by the user). Moreover, an algorithm for approximating the minimum weight Steiner triangulation, that was given by Eppstein [Epp92].

Bern et al. [BEG94] presented triangulations with triangles of bounded aspect ratio and triangulations with triangles without obtuse angles, such that the number of triangles is within a constant of the optimal number.

In addition to these algorithms there exist heuristically obtained meshes, that do not allow for a guaranteed quality (see Bern and Eppstein [BE92] for a brief overview).

In general, in case the number is considered, the focus in meshing is on the number of triangles, rather than on the number of relays. Moreover, the resulting triangles for our problems may have arbitrary angles, aspect ratio, height and also the total edge length sum of triangles is not in the focus of our interest. Hence, while the use of Steiner points and the interest in a special shape of the triangles is similar, the optimization criteria for mesh generation and for the MRTP and the MATP differ. To the best of our knowledge the optimization criteria we consider have not been treated for a meshing problem.

Relay Placement. Given a set of sensors, the network should be connected by placing additional nodes, relays. Sensors and relays have a limited communication range. The goal is to use as few relays as possible. In contrast to the MRTP and the MATP that ask for a triangulated structure, for the relay placement problem the sensors need to be connected via paths (those connecting paths may be allowed to contain sensors (one-tier version) or not (two-tier version)). This problem has been considered by Efrat et al. [EFG$^+$08] who presented a 3.11-approximation for the one-tier version and a PTAS for the two-tier version.

Dispersion. The problem of dispersing a swarm of robots into a cellular environment was considered by Hsiang et al. [HAB$^+$02, HAB$^+$03]. They aim for minimizing the time until each cell of the environment is occupied by a robot.

Dispersing a swarm of robot has also been considered from a practical point of view. Such a practical dispersion problem was, for example, solved by McLurkin and Smith [MS07], who dispersed a group of autonomous mobile

robots.

Orienteering. Consider a graph with length on edges and rewards on vertices. Moreover, the explorer is provided with a hard limit on the cost, the sum of length of traversed edges. One might think of a limited battery. In the orienteering problem, the goal is to find a path or a tour or a tree that does not exceed the cost limit and that maximizes the total collected reward. For the rooted orienteering problem a start point is given. Arkin et al. [AMN98] were the first who presented constant-factor approximations for the rooted orienteering problem for the special case of points in the plane. Thus, their approximation results uses the underlying geometry. The results are based on m-guillotine subdivisions. For the general problem Blum et al. [BCK+07] gave a constant-factor approximation algorithm.

A related problem, the *k-MST* problem was considered by Awerbuch et al. [AABV98]. This problem asks for a tree that spans k out of n vertices in a graph, such that the total cost of the tree is minimized.

Geometric Knapsack Problems. The problem of finding a maximum area (or maximum perimeter) convex k-gon using k out of n given points was considered by Boyce et al. [BDDG82]. Here the limit is also—as in the MATP—on the number of points, and the area is to be maximized. But, the points to choose from are given. Boyce et al. presented a linear space and $O(kn \lg n + n \lg^2 n)$ time algorithm for this problem. Actually, all points selected for the k-gon lie on the convex hull of the point set, for both the area and perimeter version. The statement of geometric knapsack problems is more general: A subset of given objects, each with a reward, should be enclosed by a closed curve (packed in a "knapsack"), such that a constraint on the knapsack's capacity is not violated. The objects can be points, lines, polygons or the like. Arkin et al. [AKM93] considered the problem with the capacity to be defined as the curve's perimeter or area. There exist variants of the problem with a given (finite) upper bound on the capacity. For other variants the capacity is unlimited, but each unit of capacity infers a certain cost. Arkin et al. showed that for the case of rewards that are allowed to be negative, and for the case of an upper bound on the perimeter or area, the problems are NP-hard. For the variant with a cost caused by each curve unit (but without an upper bound on its length) they presented polynomial time algorithms for different object types. For all these variants the optimal enclosure is convex or, for example, for polygonal objects, geodesically convex. So while these problems also treat the connection between covered area and number of points, the non-convexity of the enclosed area for MATP as well as the not predefined objects sets clearly change the problems structure.

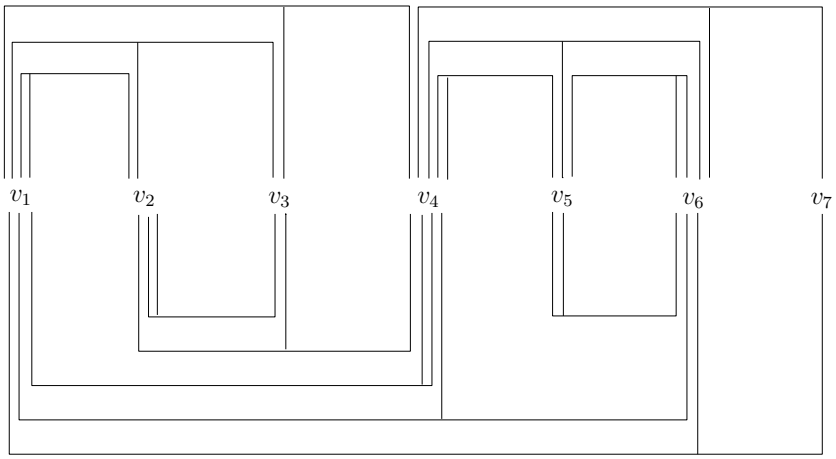


Figure 5.2: An example for a rectilinear configuration of the planar 3SAT.

5.2 NP-Hardness

In this section we show that MRTP and MATP are NP-hard.

Theorem 12. *The Minimum Relay Triangulation Problem (MRTP) is NP-hard, even without a discrete set of candidate locations.*

Proof. The proof is again based on a reduction of the NP-hard problem `Planar 3SAT`, a special case of 3SAT in which the variable-clause incidence graph H is planar. In particular, there is a rectilinear embedding of H where variables $v_1, \ldots, v_n$ lie on a straight line and the clauses are arranged above and below them as shown in Figure 5.2; see Knuth and Raghunathan [KR92].

We will turn this embedding into a polygon: We represent the variables, clauses and edges by polygonal pieces. A sketch of this transformation can be seen in Figure 5.3, the polygonal pieces are here depicted as lines with bold lines for the edges; the clause gadget is indicated in green.

An example for the clause component is given in Figure 5.4: The edge corridors, which we will describe later, of three variables end in the triangular gadget. The boundary is shown in black; all other lines are used only to highlight certain distances. The light gray triangle has a side length of $(1 + \varepsilon)$, the three dashed triangles all have side length 1. A relay placement corresponding to a truth setting that satisfies the clause (i.e., a setting of "true" if v_i is in the clause, a setting of "false" if $\neg v_i$ appears in the clause) is indicated in blue, a setting not satisfying the clause in red. The red circles are unit circles depicting the range in which the setting that does not satisfy the clause is able to establish edges. For the zigzagged boundary, the relays at the vertices have distance less than 1 (i.e., the distance between v_1 and v_2

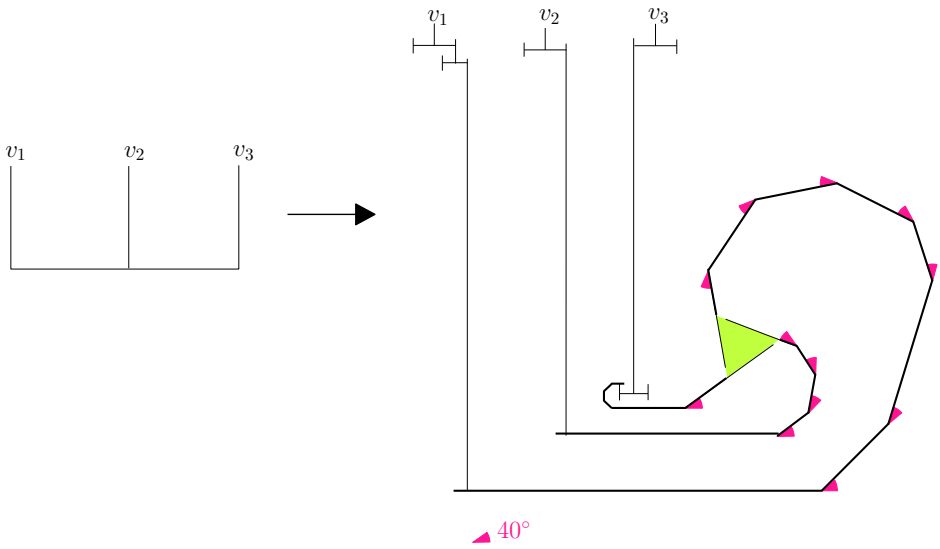


Figure 5.3: Sketch of the transformation from the rectilinear configuration of planar 3SAT: a clause with the three variables v_1, v_2, v_3 $(v_1 \vee v_2 \vee \neg v_3)$. The polygonal pieces are here shown as lines; with bold lines for the edges, and the clause gadget is indicated in green.

is less than 1, as is the distance between v_3 and v_4 and the distance between v_5 and v_6); thus, an additional relay is needed only between the last relay from the corridor (blue or red) and the next vertex (of the connection to a corridor relating to a different variable), for example, between v_2 and p_3 or p_4—yielding a total of 3 relays. Consequently, we need to consider the relays required in the component's interior. In case no variable satisfies the clause, interior relays inside of each of the red circles need to be added. But we cannot place a triangle with side length (at most) 1 with vertices inside these circles. This fact is illustrated in Figure 5.5: The violet lines indicate the edges of length less than 1 that are directly established by the relays corresponding to a truth setting not satisfying the clause. For example, the upper right red relay placed at p_4 cannot connect with the (light green) relay at v_4. Moreover, this light green relay at v_4 cannot be connected to any relay placed on the same boundary edge as the red relay. Hence, 3 additional interior relays are not sufficient for a complete triangulation; that is, in case no variable satisfies the clause, 4 interior relays are required for the clause component.

For the case that at least one variable satisfies the clause, see Figure 5.6, with relays placed at p_1, p_4 and p_6: The relays relating to the corners of one of the dashed inner triangles, here the pink one, at r_1, r_2 and r_3, are sufficient to establish a complete triangulation. In case more than one variable satisfies

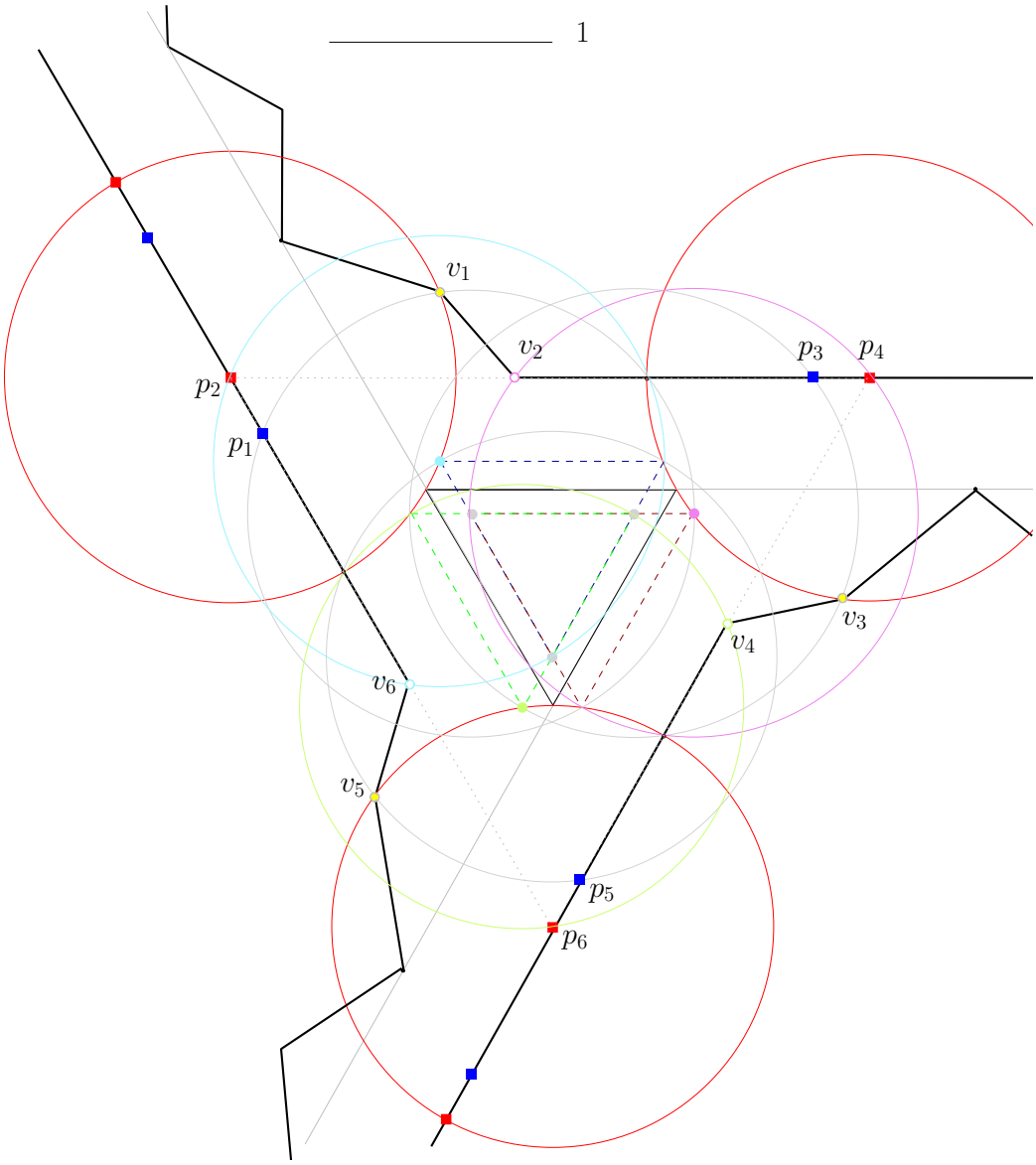


Figure 5.4: The polygonal piece for a clause.

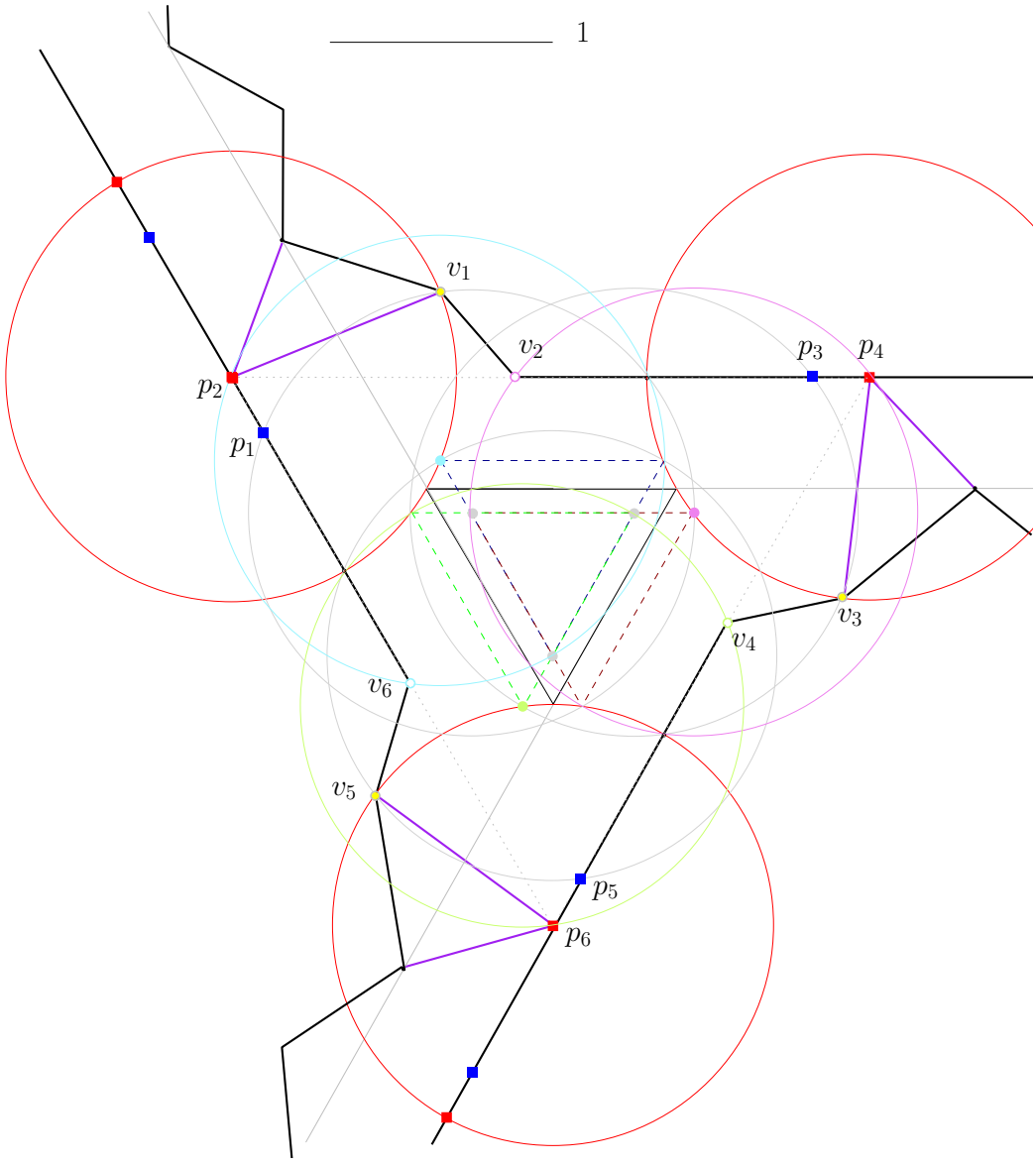


Figure 5.5: The polygonal piece for a clause. The violet lines indicate the edges established from the relay positions corresponding to variables not satisfying the clause.

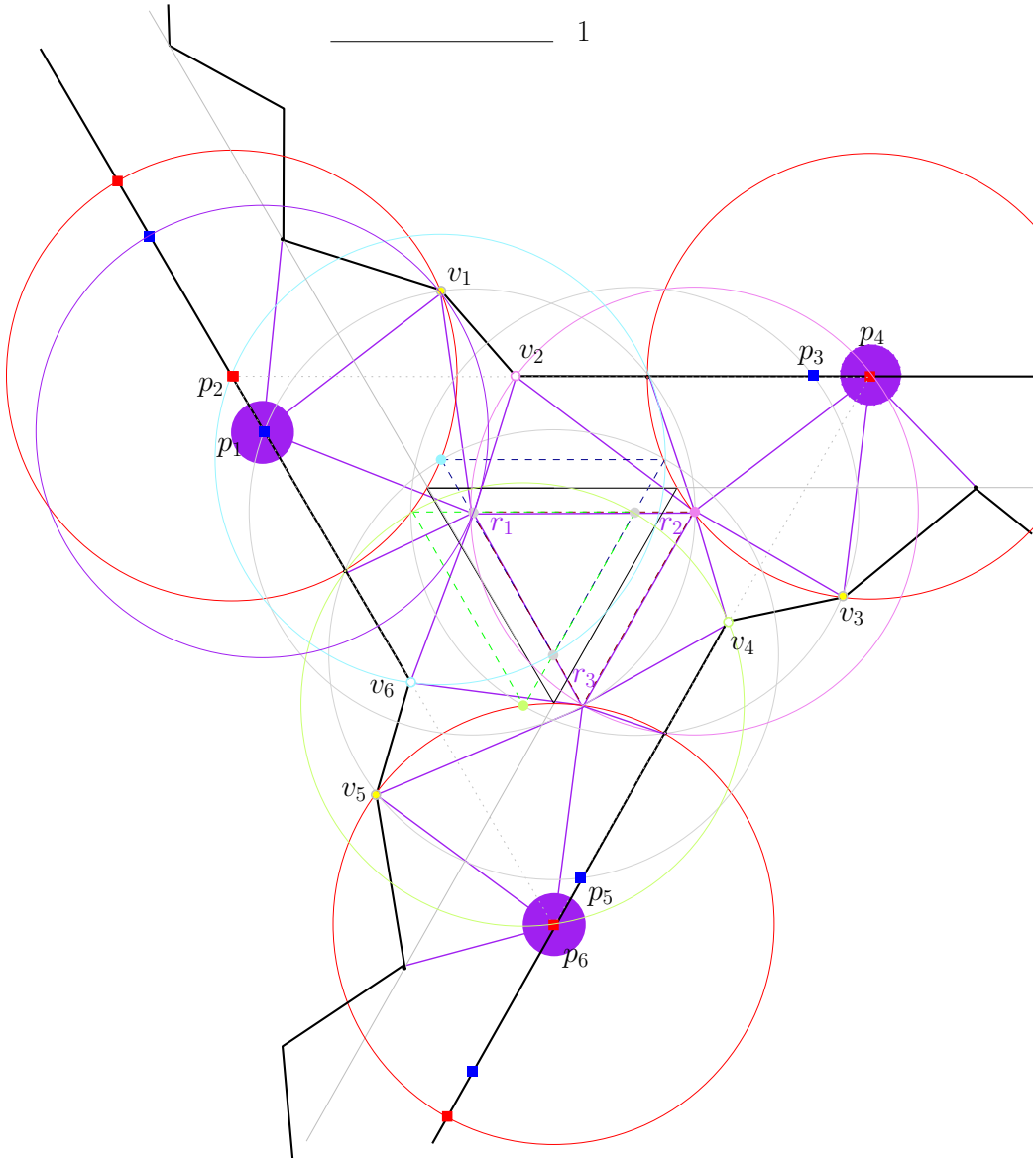


Figure 5.6: The polygonal piece for a clause. An exemplary truth setting is highlighted in violet: The variable corresponding to the upper left corridor satisfies the clause, with a relay placed at p_1, the other two variables do not (relays placed at p_4 and p_6). The resulting triangulation is shown with violet lines.

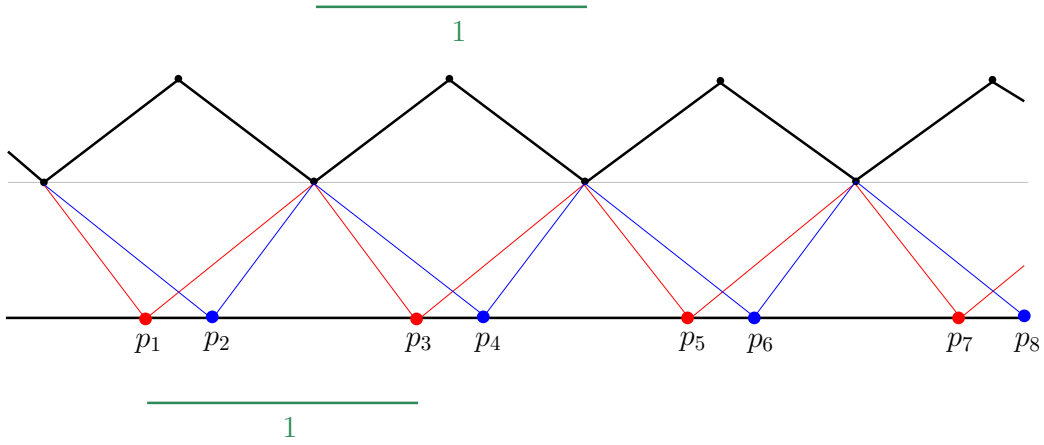


Figure 5.7: The polygonal piece for the edges.

the clause, such an interior triangulation is obviously still possible. Thus, in case at least one variable satisfies the clause, 3 interior relays are sufficient for the clause component.

An example for the edge component is given in Figure 5.7. The corridor is bounded by a straight line on one side and a zigzagg line on the other. The zigzagg line is constructed as follows: We place vertices in distance 1 from each other on a line parallel to the straight boundary of the corridor in distance $\frac{1}{2}$. Between those vertices we place "peak" vertices in distance at least 1 from the parallel line. The relays placed on the vertices of the zigzagg boundary (indicated in black) suffice to triangulate the peaks. On the long edge of the boundary, relays can be placed in distance of exactly 1 (such that the distances to both lower peak relays is also less than 1 as indicated by the red and blue lines). Thus, considering the edge from left to right, a "pushed" relay (i.e., a placement of relays of distance 1 on the straight line) that starts further inside the corridor can be "pushed" further to the right than a "pushed" relay initially located more to the left. In our construction, we will push the truth setting satisfying the clause further in the corridor, and in the end further into the clause component. (Consequently, considering Figure 5.7, we see that a relay located at the leftmost red point p_1 allows only for a location of the fourth relay on the rightmost red point p_7. While, starting with a relay located at the leftmost blue point p_2, the fourth relay may be positioned on the rightmost blue point p_8.)

When considering the overview of the construction in Figure 5.3 we observe that it is not sufficient to have one straight corridor starting at the variable component and ending at the clause component, but we need to be able:

1. to make 45°-bends with these corridors (in the direction of the triangles) in order to be able to connect to the corridors in case of a negated variable (we need to bend the corridors by 180°) and

2. to make 40°-bends with these corridors (in the direction of the triangles)– we need to bend the corridors by 120°–and

3. to shorten and to prolong the distances in the triangular structure and

4. to shorten and to prolong the triangular structure.

In order to do so, we will demonstrate the constructions and consider feasible distances (e.g., of a point that is to be "pushed" through a corridor to the interval bounds determined by the triangle relays). We will see that there are distances that are feasible for the entire construction.

We start with the third item: shortening or prolonging the triangular structure and distances in this structure. Let d_A be the initial distance of points according to the two truth settings; we want to end up with a distance of d. The general construction of such an extension/shortening-piece is shown in Figure 5.8, the resulting triangulation for both truth settings is given in Figure 5.9. The left blue point at p_2, corresponding to a truth setting satisfying the clause, allows for an edge to the left upper vertex v_2 (denoted by a green cross). In contrast, using the red point at p_1 (corresponding to a truth setting not satisfying the clause) an additional relay on the left edge to this vertex v_2 is needed. As we may spend an equal number of relays, the clause-satisfying setting allows for an additional relay on the edge incident to the second upper vertex (v_3, v_4) (highlighted in yellow), enabling a position for the rightmost blue point at p_4 more to the right than for the rightmost red point at p_3.

Given this general construction, we need to consider feasible distances. See Figure 5.10: For the left part (Figure 5.10 (a)) with given distances d_A in between the two points p_1 and p_2 corresponding to the different truth settings and $1 - d_l$ as the horizontal distance from the point corresponding to the truth setting not satisfying the clause, p_2, to the left border of a triangular structure of length 1, p_0, the feasibility constraint is $\ell_x < 1$. We construct the intersection point, p_{int1}, of the two circles of radius 1 around p_2 (blue) and around v_1 (black). Then, we draw a line from p_{int1} to v_1, the last vertex of the left triangular structure. The distance from v_1 to the intersection of the drawn line with a circle of radius 1 around p_1 is defined as ℓ_x. Thus, starting with a relay at p_1, corresponding to a truth setting not satisfying the clause, an additional relay has to be placed along that line; starting with a relay at p_2 a triangle including $p_{\text{int1}} = v_2$ can be constructed.

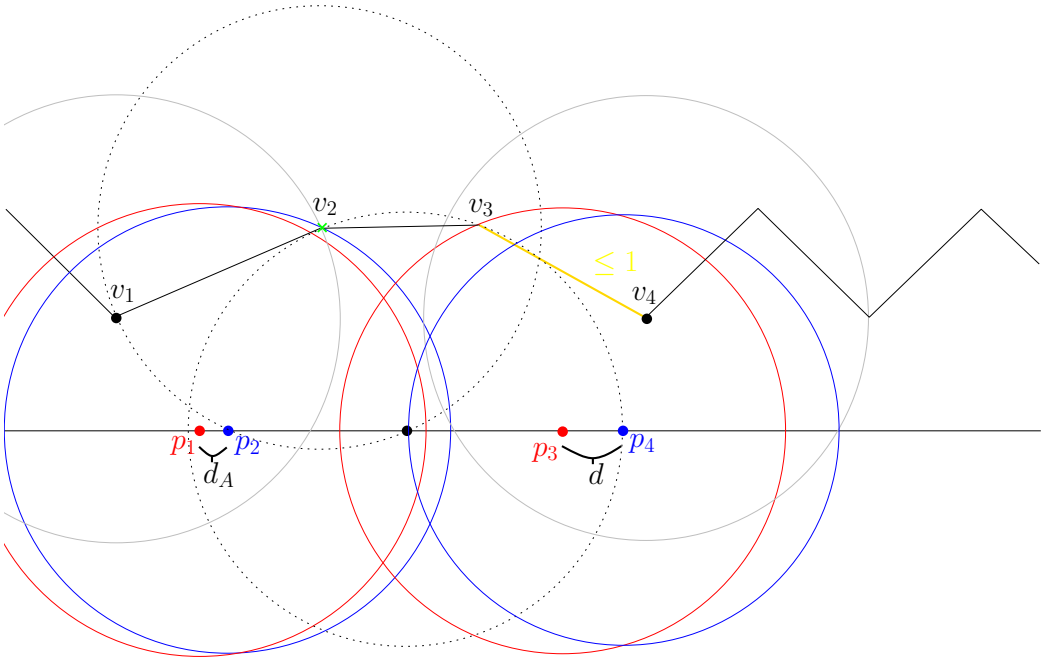


Figure 5.8: Construction for changing the distance of the two points corresponding to the two truth settings (blue and red) from d_A to d.

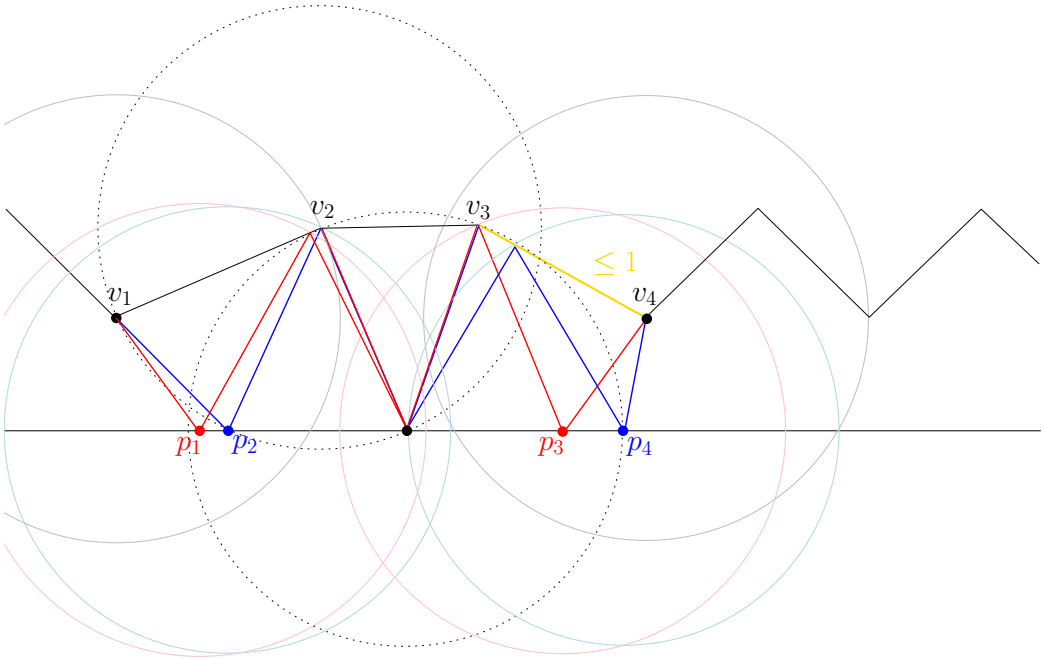


Figure 5.9: The resulting triangulation for the polygonal piece from Figure 5.8.

Using the notation from Figure 5.10 (a), we have:

$$a = \sqrt{\frac{1}{4} + (1 - d_l)^2}, b = \sqrt{\frac{1}{4} + (1 - d_l + d_A)^2} \tag{5.1}$$

$$\beta = \arccos(\frac{b}{2}), \gamma = \arccos(\frac{a^2 + b^2 - d_A^2}{2ab}) \tag{5.2}$$

$$\alpha = \arcsin(a \cdot \sin(\beta + \gamma)) \tag{5.3}$$

$$\delta = \pi - \alpha - \beta - \gamma \tag{5.4}$$

$$\Rightarrow \ell_x = \sqrt{1^2 + a^2 - 2 \cdot a \cdot \cos(\delta)} \tag{5.5}$$

Thus, the feasibility constraint is:

$$\begin{aligned} 1 > \ell_x \quad = \quad & \sqrt{\frac{5}{4} + (1 - d_l)^2 - 2\sqrt{\frac{1}{4} + (1 - d_l)^2} \cdot \cos\left(\pi - \right.} \\ & \overline{\arcsin\left(a \cdot \sin\left(\arccos\left(\frac{\sqrt{\frac{1}{4} + (1 - d_l + d_A)^2}}{2}\right)\right.\right.} \\ & \overline{\left.\left. + \arccos\left(\frac{\frac{1}{4} + (1 - d_l)^2 + \frac{1}{4} + (1 - d_l + d_A)^2 - d_A^2}{2 \cdot \sqrt{\frac{1}{4} + (1 - d_l)^2} \cdot \sqrt{\frac{1}{4} + (1 - d_l + d_A)^2}}\right)\right)\right)} \\ & \overline{- \arccos\left(\frac{\sqrt{\frac{1}{4} + (1 - d_l + d_A)^2}}{2}\right)} \\ & \overline{\left. - \arccos\left(\frac{\frac{1}{4} + (1 - d_l)^2 + \frac{1}{4} + (1 - d_l + d_A)^2 - d_A^2}{2 \cdot \sqrt{\frac{1}{4} + (1 - d_l)^2}}\right)\right)} \end{aligned} \tag{5.6}$$

For the right part (Figure 5.10 (b)) with given distances of d in between p_3 and p_4 (the two points corresponding to the different truth settings) and d_r as the horizontal distance from p_4 to the right border of a triangular structure of length 1, p_r, the feasibility constraint is $\ell \leq 1$: We construct the intersection point, p_{int2}, of the two circles of radius 1 around p_3 and around the central point p_m (in distance 1 to p_4, the point corresponding to a clause satisfying truth setting). Then, we draw a line from p_{int2} to the first vertex of the right triangular structure, v_4. ℓ is the length of this line. For the truth setting that does not satisfy the clause, we already used the additional relay for the triangulation to the left. Thus, the red point at p_3 is the rightmost feasible location for another relay in this case. For the truth setting satisfying the

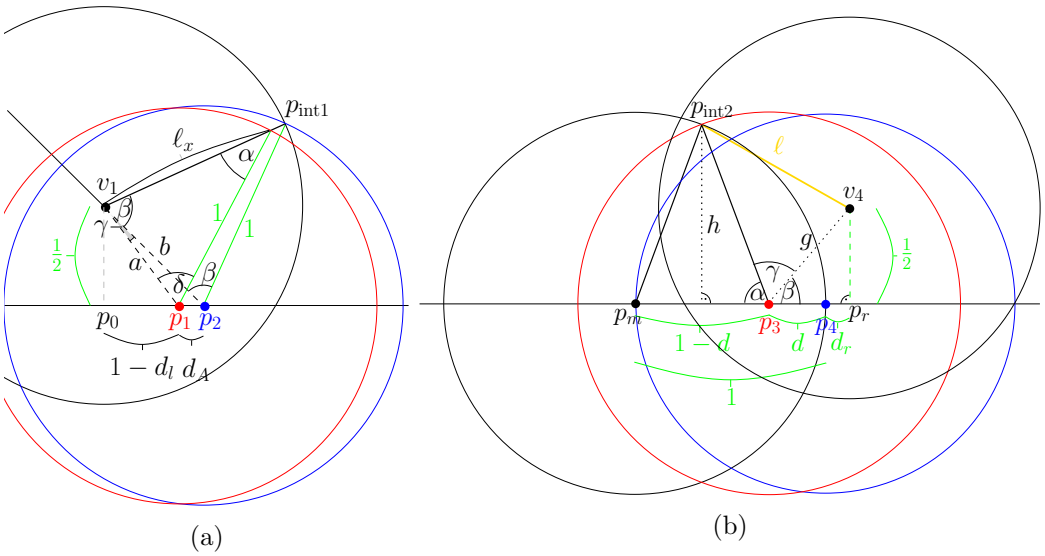


Figure 5.10: (a) $\ell_x < 1$. (b) $\ell < 1$.

clause, we may spend an additional relay along the constructed line, allowing for a relay positioning at the rightmost blue point p_4 (within distance 1 to the central point p_m).

Using the notation from Figure 5.10 (b), we have:

$$h = \frac{1}{2}\sqrt{3+2d-d^2}, g = \sqrt{\frac{1}{4}+(d+d_r)^2} \tag{5.7}$$

$$\alpha = \arcsin(h), \beta = \arcsin\left(\frac{1}{2g}\right) \tag{5.8}$$

$$\gamma = \pi - \alpha - \beta \tag{5.9}$$

$$\ell^2 = 1^2 + g^2 - 2 \cdot g \cdot \cos(g) \tag{5.10}$$

Thus, the feasibility constraint is:

$$1 \geq \ell = \frac{\sqrt{\frac{5}{4}+(d+d_r)^2 - 2 \cdot \sqrt{\frac{1}{4}+(d+d_r)^2}\cdot}}{\cos\left(\pi - \arcsin\left(\frac{1}{2}\sqrt{3+2d-d^2}\right)\right.} \\ \left. - \arcsin\left(\frac{1}{2\sqrt{\frac{1}{4}+(d+d_r)^2}}\right)\right) \tag{5.11}$$

Moreover, we have to ensure that we can bend the corridors by 40° and 45°. We start with a 45°-bend of the corridors. The general construction

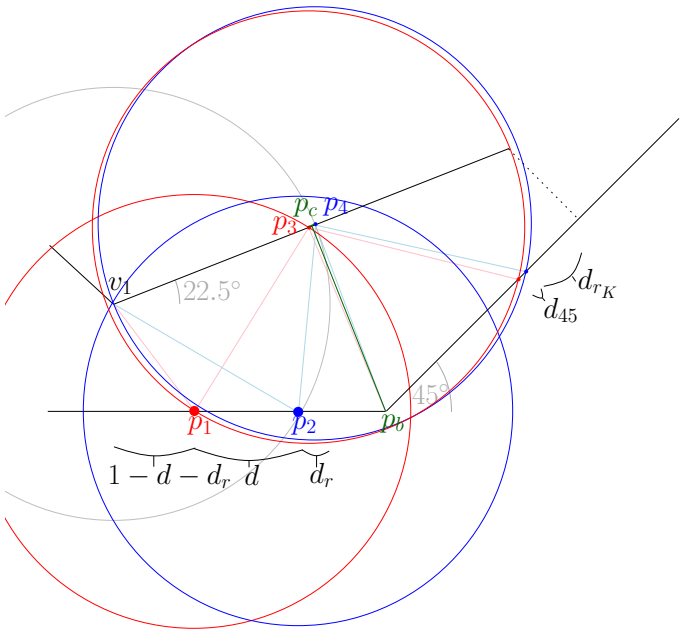


Figure 5.11: Construction of a 45°-bend of a corridor.

is shown in Figure 5.11. We start with a red point and a blue point, at p_1 and p_2, corresponding to a truth setting not satisfying the clause and a truth setting satisfying the clause again, respectively. Given distances d and d_r, we draw a line with a slope of 22.5° from the last vertex of the triangular structure, v_1. Consider the two points where circles of radius 1 around v_1 and around p_1 (depicted in grey and red) intersect the line of slope 22.5°. From the center point between these points, p_c, we imagine an orthogonal line (shown in dark green). At the point where this line meets the bottom edge, p_b, we construct a 45°-bend of the boundary. The resulting triangulations for both truth settings is indicated in Figure 5.12 (notice that the second blue point p_4 is not located on the blue circle but within distance 1 to the last vertex of the triangular structure).

The location of the red point is feasible for

$$1 - d - d_r < \cos\left(\frac{\pi}{8}\right) - \sqrt{1 - \left(\sin\left(\frac{\pi}{8}\right) + \frac{1}{2}\right)^2} \tag{5.12}$$

(For $x = 1$ we have $1-d-d_r = \cos\left(\frac{\pi}{8}\right) - \sqrt{1 - \left(\sin\left(\frac{\pi}{8}\right) + \frac{1}{2}\right)^2}$.) The location of the blue point is feasible for

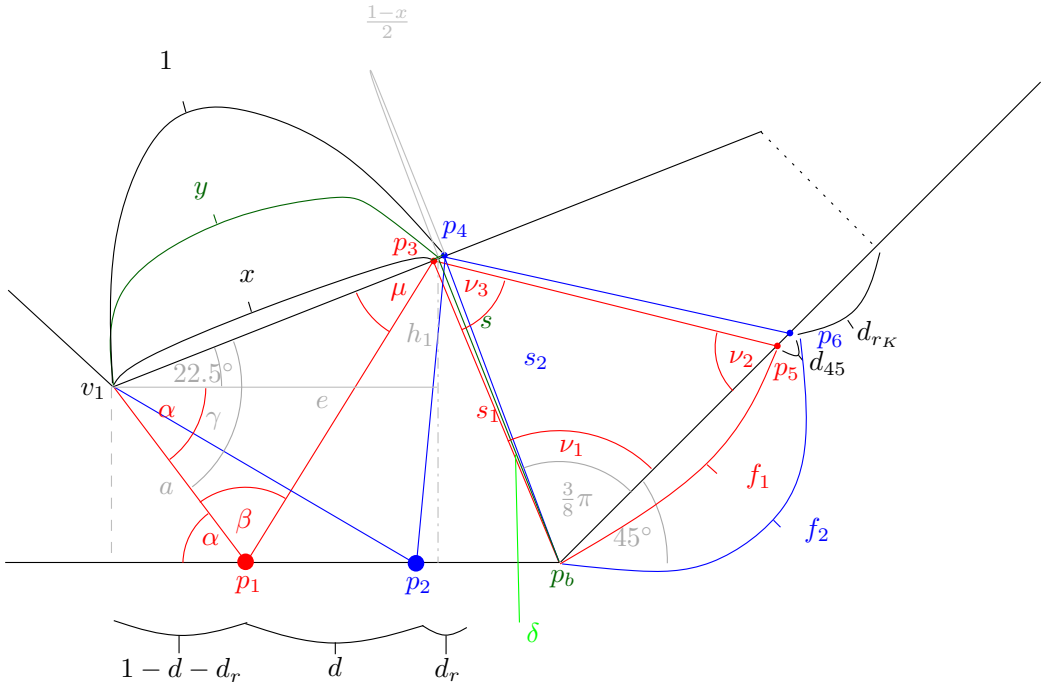


Figure 5.12: 45°-bend with labels for angles etc. which allow for the computation of the distances.

(c1) $1 - d_r \geq \cos\left(\frac{\pi}{8}\right) - \sqrt{1 - \left(\sin\left(\frac{\pi}{8}\right) + \frac{1}{2}\right)^2}$ and

(c2) $1 - d_r \leq \frac{\sqrt{3}}{2}$

In order to have a feasible input for the 45°-bend, which we may construct using the polygonal piece for extending or shortening the distances (Figure 5.8), we search for a pair (d, d_r) meeting conditions (5.11), (5.12), (c1) and (c2). The tuple $(d, d_r) = (0.45, 0.15)$ is such a point. As we have continuous functions, there is an ε-neighborhood around this point that is feasible.

So, the last thing we need to consider regarding the corridors are the distances d_{45} and d_{r_K} (as defined in Figure 5.12). The resulting question is: Is there a feasible input to the bend that produces an output that is in turn a feasible input for the extension/shortening-piece. We can determine d_{45} and d_{r_K} in dependence of (d, d_r). As $l_x[d_{45}(0.45, 0.15), d_{r_K}(0.45, 0.15)] \approx 0.980439 < 1$ such a feasible input with a feasible ε-neighborhood exists.

Analog computations yield the feasiblilty of 40°-bends of the corridors (see Figure 5.13). Note that, for example, 30°-bends cannot be constructed (which would allow for both the necessary 120° and 180° bends).

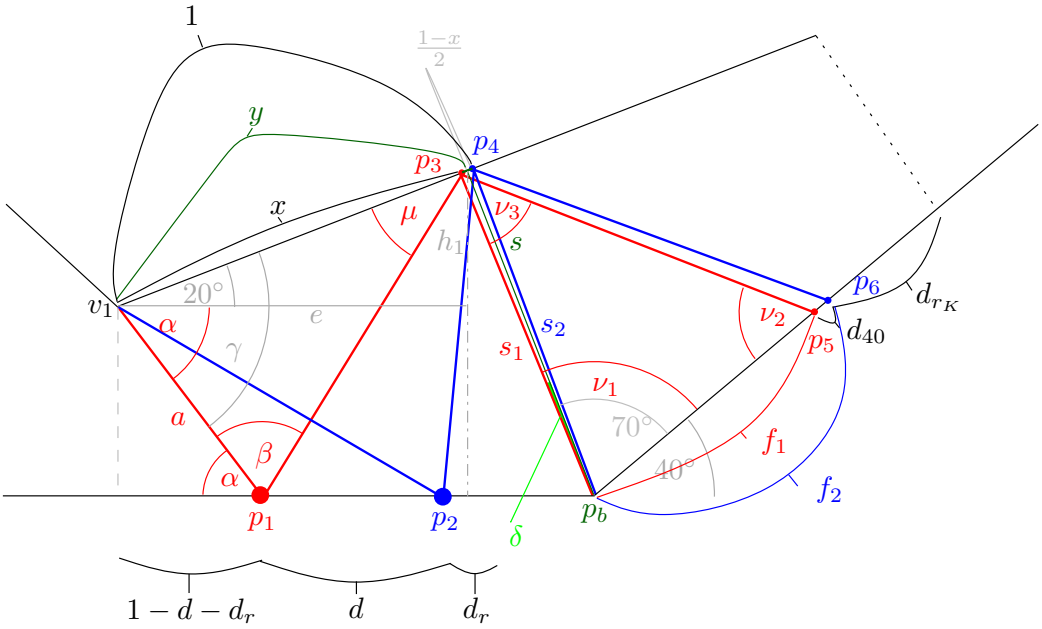


Figure 5.13: Construction of a 40°-bend of a corridor.

Up till now we changed the distances in the triangular structure. We need to make sure that we may use the same input and output distances while shortening or extending the distance between the triangles–enabling us to give the overall construction. Consider Figure 5.14: Using a feasible setup (e.g., $d = d_A = \frac{1}{4}, d_r = \frac{1}{4}$), the central point, p_m, may be located anywhere within a distance d_c with $d < d_c \leq 1$ (on the green line). Possibly using several of these constructions, we may obtain any extension or shortening.

Finally, we need to consider the variable component. An example is given in Figure 5.15. The (horizontal) baseline of the construction is slightly longer than 1. This baseline is depicted in bold in the upper right of Figure 5.15. Vertical edges are attached. The vertices at the ends of these vertical edges may be reached only by a triangulation starting with a relay left or right from the center of the baseline. See the dark gray dashed and the light gray triangulation in Figure 5.15: The light gray triangulation needs two relays on the left vertical edge, and three on the right one, the dark gray one vice versa. In the same distance as started, that is, as placed on the baseline, a dark gray and a light gray point is pushed into the horizontal polygonal piece. The width of this horizontal polygonal piece is determined by the vertical distance of the pushed points to the vertices plus $\frac{1}{2}$. Here again the construction is such that the dark gray point ends up to the right of the light gray point in both attached vertical polygonal pieces. (In Figure 5.15 only the left of the two vertical pieces that may be attached to the first vertical

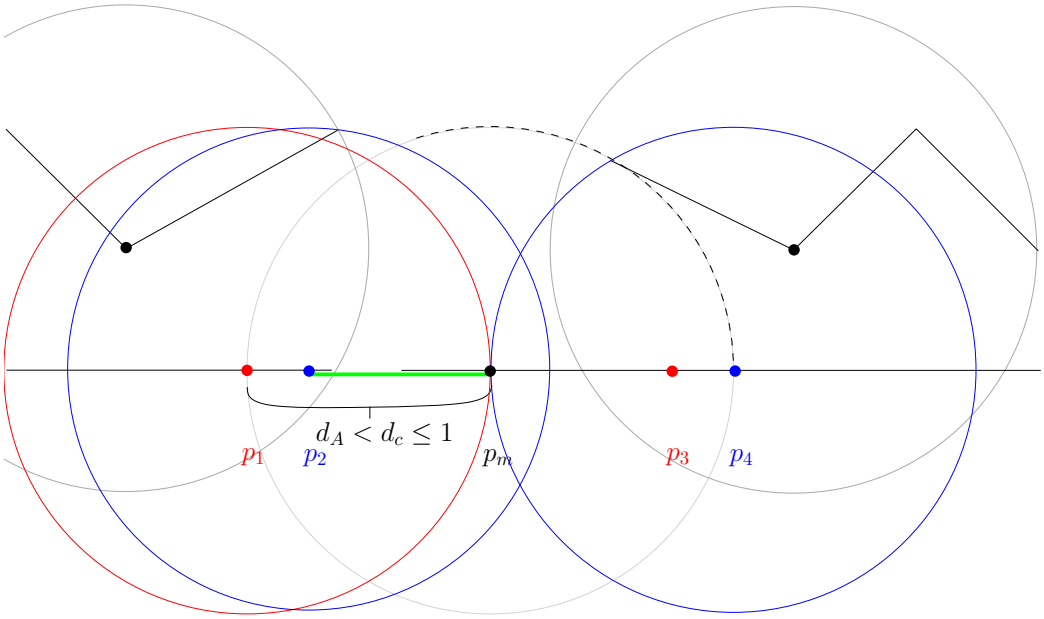


Figure 5.14: Polygonal piece for changing the distance between the triangles.

polygonal piece with the baseline is shown.) The light gray dotted lines indicate the triangulations in case a polygonal piece is closed, such that both triangulations, corresponding to a truth setting satisfying the clause and a truth setting not satisfying the clause, need the same number of relays. Let us consider variable v_i: In case v_i appears in a clause, an edge corridor is attached at the lower side of one of the vertical polygonal pieces, as in the green ellipse in Figure 5.15. Thus, the dark gray point is "pushed" further into the edge corridor. If $\neg v_i$ appears in the clause, an edge corridor is attached at the upper side of one of the vertical polygonal pieces, as in the blue ellipse in Figure 5.15. Thus, the light gray point is "pushed" further into the edge corridor. For each horizontal polygonal piece, we increase the number of vertical polygonal pieces that allow for a connection of an edge corridor. Consequently, this construction allows for the connection of an arbitrary number of edge corridors leading to the corresponding clause gadgets. A feasible configuration is (d, d_r) with $d+2(1-d_r) < 1$, e.g., $d = 0.1, d_r = 0.4$ resulting in a baseline of length 1.1. (For the vertical corridors the corresponding mirrored triangulation is also feasible.)

Finally, we check that the input distance for the clause gadget (Figure 5.4) is a feasible output of our corridor edges. We have $l(0.22, 0.25) \approx 0.957 < 1$, l being a continuous function we thus have a feasible ε-neighborhood around this point.

Given the components defined above we can compute the parameter k,

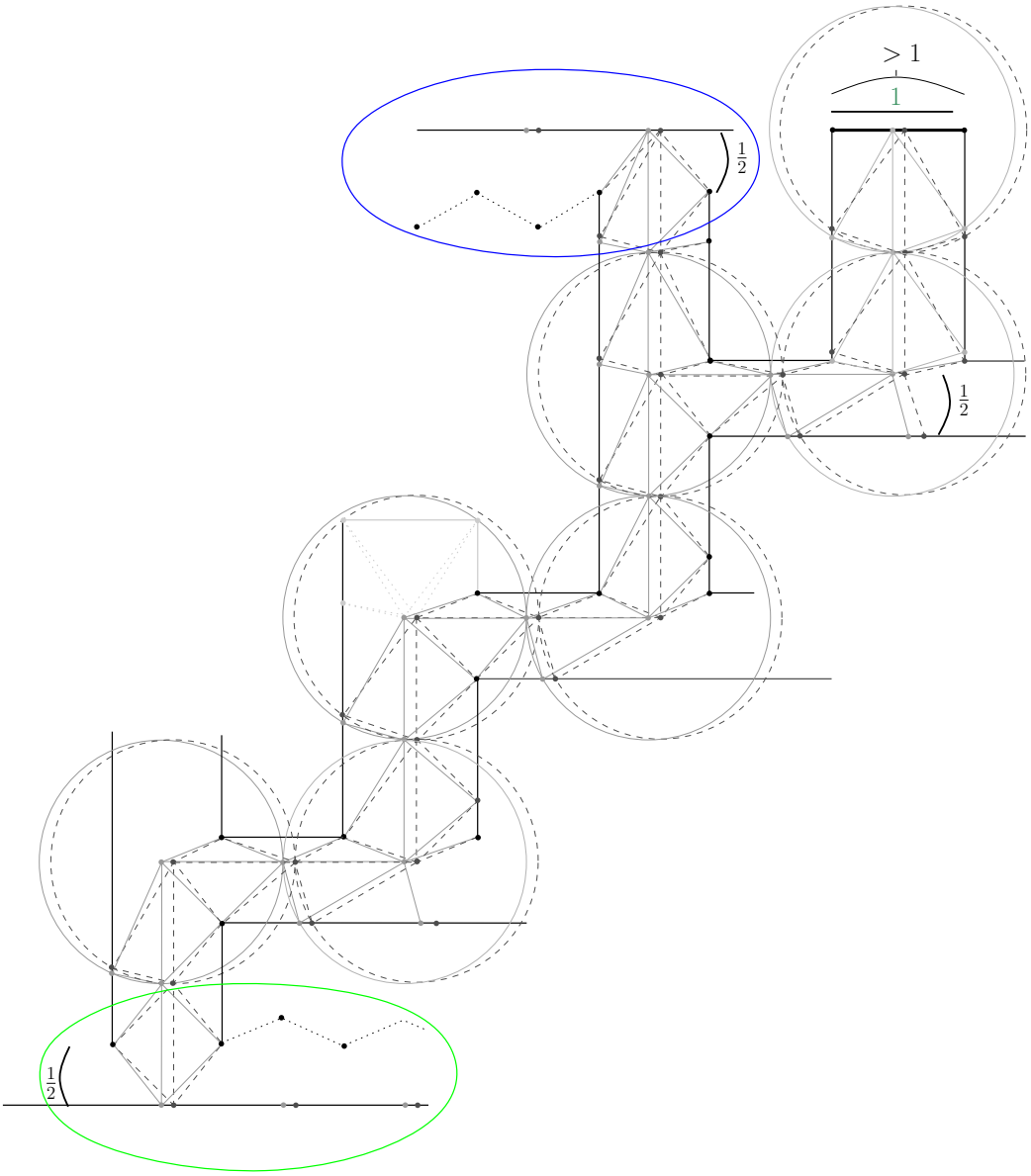


Figure 5.15: The polygonal piece for the variable.

the number of relays necessary to triangulate the entire resulting polygon P. k is polynomial in the number of vertices of H and part of the input. Finally, we need to have a look at the coordinates: In the variable gadgets and within the edge corridors up to the first 40°-bend, we have rational coordinates (in the variable gadgets we did choose rational coordinates above, in the corridors we keep distances of 1 along the x-axis). Within the structure surrounding the clause gadget (which is the same for all clauses), no more than 7 bends appear, thus we may perturb the coordinates by an ε in order to have rational coordinates, such that there is an error in the distances between the points corresponding to the different truth settings. This error is propagated; but, having only $O(1)$ such changes this error can be kept below any constant.

□

5.3 Online Minimum Relay Triangulation

In Section 5.2 we analyzed the complexity of MRTP and MATP—we proved both the MRTP and the MATP to be NP-hard. So, we have a classification of the hardness of the offline problem. But, in particular for the application, our focus is on the online variants rather than the offline variants. We intend to expand a given sensor network as a reaction to a new situation. It is likely that we will not be equipped with a complete map of the new environment. Thus, in this section we consider the OMRTP, and we study the OMATP in Section 5.4. We establish a lower bound of 6/5 on the competitive ratio for any online strategy for the OMRTP and we present a 3-competitive online algorithm for the OMRTP. Note that the online strategy also provides a 3-approximation algorithm.

5.3.1 Lower Bound

For the lower bound we use a polygonal corridor of width 3/4. For a complete triangulation, relays must be placed at the vertices, i.e., the position of the first two relays is fixed.

In case the algorithm places the next relay on the right boundary, the polygonal corridor will turn out to look like in Figure 5.16(a). We need to determine the number of relays up to the two relays connected by the dotted edge (in the area indicated by the light gray shape in Figure 5.16), those build the two fixed relays of the next polygonal piece. The optimum needs 5 relays. The distance of the relay placed by the algorithm on the right boundary to the next vertex is larger than 1, thus, the algorithm uses 6 relays, see Figure 5.16(b).

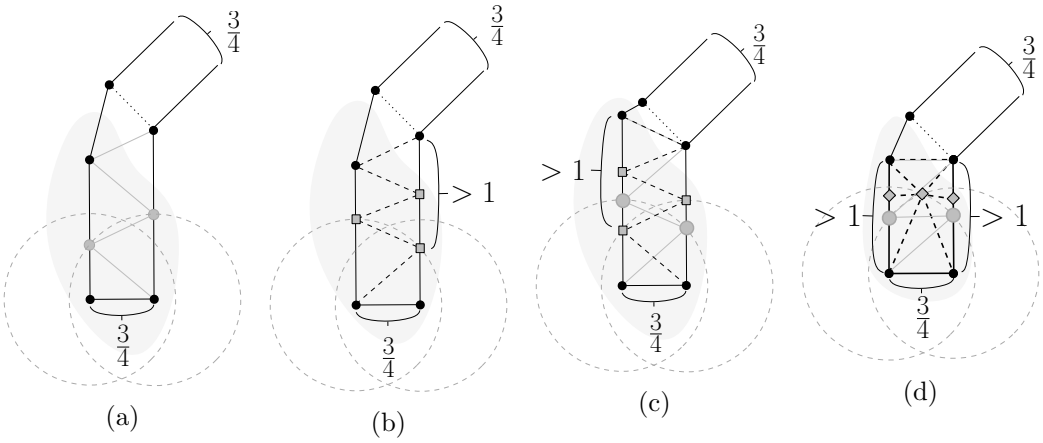


Figure 5.16: A lower bound for the OMRTP. Black dots indicate polygon vertices, i.e., mandatory relays; grey disks indicate an optimal solution, while grey squares indicate relays placed by an online strategy.

In case the algorithm locates the next relay on the left boundary, the polygonal corridor turns out to look like in Figure 5.16(c), and the optimum locates 5 relays, while the algorithm must place 6 relays.

If, on the other hand, the algorithm places the next relay in the center, the polygonal corridor turns out to look like in Figure 5.16(d), again with an optimum of 5, and an online solution of 6.

The construction we presented results in the next component connected in 45° to the right. Constructions for a connection within a 45° angle to the left are done analogously (mirrored constructions)—resulting in reflected components.

The additional relays ensure that a final corridor of 3/4 is achieved again. Thus, we can iterate the construction. We alternate between the components shown in Figure 5.16 and the reflected components to avoid a self-overlapping polygon.

Altogether, we obtain a bound on the competitive ratio for any deterministic algorithm for the OMRTP.

Theorem 13. *No deterministic algorithm for the online minimum relay triangulation problem can be better than* $\frac{6}{5}$*-competitive.*

5.3.2 Online Triangulation

In this subsection we present our online algorithm for the relay triangulation problem. For our construction we use two components that we combine into one triangulation using a third set of relays. The two components are (1) a placement along P's boundary and (2) a triangulation of P's interior.

For (1) we place relays within distance 1 along the boundary and on vertices; when interior boundaries are encountered they are treated in the same way. Let b_{ALG} be the number of relays used in this step, and b_{OPT} the number of relays placed on the boundary by an optimal solution. Any triangulation must establish triangulation edges along all edges of the polygon P. As the maximum distance of relays that allow for a connecting edge is $r = 1$, we conclude

Lemma 7. $b_{\mathrm{ALG}} \leq b_{\mathrm{OPT}}$.

For the triangulation of P's interior, that is, step 2, we overlay the polygon's interior with a grid: From the starting point we construct an arbitrarily oriented triangular unit grid. Whenever we cannot continue this grid but are able to place a relay with distance 1 to all existing interior relays, we do so (and resume the grid construction when possible). Let i_{ALG} be the number of relays used in this step.

To achieve an upper bound on i_{ALG}, we consider the number of triangles in an optimal solution.

Lemma 8. *For an optimal solution for the MRTP with b_{OPT} relays located on the boundary and i_{OPT} located in the interior, the number $\triangle_{\mathrm{OPT}}$ of triangles in an optimal triangulation satisfies $\triangle_{\mathrm{OPT}} = 2 \cdot i_{\mathrm{OPT}} + b_{\mathrm{OPT}} - 2$.*

Proof. We consider the sum of interior angles. For the boundary we have $\pi(b_{\mathrm{OPT}} - 2)$ as the sum of interior angles. At each interior point an additional 2π occurs. As we have a share of π for each triangle, we obtain

$$\triangle_{\mathrm{OPT}} = \frac{\pi(b_{\mathrm{OPT}} - 2) + i_{\mathrm{OPT}} \cdot 2\pi}{\pi} = 2 \cdot i_{\mathrm{OPT}} + b_{\mathrm{OPT}} - 2 \tag{5.13}$$

□

Knowing the number of optimal triangles, we give a bound on the number of interior grid relays by comparing them to the triangles.

Lemma 9. $i_{\mathrm{ALG}} \leq \triangle_{\mathrm{OPT}}$.

Proof. We charge the i_{ALG} interior grid relays to the $\triangle_{\mathrm{OPT}}$ optimal triangles. When an interior grid relay is positioned on a point that is shared by k optimal triangles, we charge a cost of $1/k$ to each of them. Thus, an interior grid relay located in a triangles' interior causes a cost of 1 assigned to this triangle. The placement of interior relays in step 2 assures a minimum distance of 1 between any two interior relays. As optimal triangles have a diameter of at most 1, a triangle will be charged by more than one relay only if it features interior relays on vertices—only on vertices. Moreover, interior relays

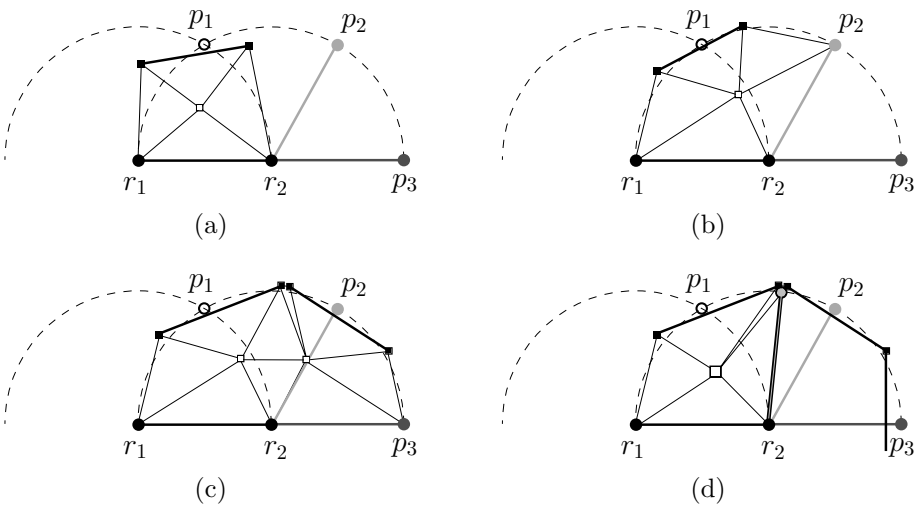


Figure 5.17: We consider the black edge $\{r_1, r_2\}$ of a triangulation. The circle position p_1 of the grid is not included. Boundary relays are denoted by black squares, relays placed in phase (ii) as black circles and relays placed to glue the triangulations by black-and-white squares. The resulting triangulations are indicated by fine black lines.

are placed strictly in P's interior (and not on the boundary). Consequently, an interior relay placed on an optimal triangle vertex is incident to at least 3 optimal triangles. Hence, also in this case, not more than $3 \cdot 1/3 = 1$ is charged to a single triangle.

To summarize: We charged a cost of at most 1 interior relay to each optimal triangle. □

Having positioned $b_{\mathrm{ALG}} + i_{\mathrm{ALG}}$ relays, we are left with the task of assuring that we combine both placements for a triangulation of the explored region. Whenever we encounter an untriangulated cell bounded by a number of connections between positioned relays, we use additional relays; let their total number be c_{ALG}. We claim that:

Lemma 10. *In total, $c_{\mathrm{ALG}} \leq b_{\mathrm{OPT}}$ additional relays suffice to ensure an overall triangulation.*

Proof. First note that interior relays of degree 1 can be triangulated without causing further cost—they can be triangulated establishing edges to boundary relays. (If not, an incident interior relay could be placed, resulting in a contradiction to degree 1.)

We consider an edge between two relays ($\{r_1, r_2\}$), see Figure 5.17, placed in (2) on the boundary of a non-triangulated cell. As position p_1 was not

occupied by a robot it must be separated from r_1 and r_2 by boundary (of P). We may not have both semicircles without boundary relays (as we may not have a line of length less than 1 with both ends outside of both semicircles). So, w.l.o.g. assume that we have a boundary edge with a relay in the left semicircle. Then:

(I) If the other relay of this boundary edge is in the right semicircle, then we have a cell of 4 relays, with one boundary edge for which we need one more relay, see Figure 5.17(a).

(II) If the other relay is not located in the right semicircle and

a) if p_2 is part of the grid, the other relay of the boundary edge is connected to this, so again we close a cell, for which we need one relay, having one boundary edge, see Figure 5.17(b).

b) if p_2 is not part of the grid and

A) if p_3 is in the triangulation, we must have another edge that separates p_2, leaving us with at least two boundary edges to which we may charge the two necessary relays, see Figure 5.17(c).

B) if p_3 is not in the triangulation we could have placed another relay with minimum distance 1 to the right black relay (drawn as a black-gray circle in, see Figure 5.17(d)), again closing the cell, for which we need one relay.

Altogether, every relay on the boundary gets charged at most once. □

Theorem 14. *There is a 3-competitive strategy for the online minimum relay triangulation problem in polygons (even with holes).*

Proof.

$$\begin{aligned} R_{\mathrm{ALG}} &= b_{\mathrm{ALG}} + i_{\mathrm{ALG}} + c_{\mathrm{ALG}} \\ &\leq b_{\mathrm{OPT}} + 2 \cdot i_{\mathrm{OPT}} + b_{\mathrm{OPT}} - 2 + b_{\mathrm{OPT}} \\ &\leq 3 \cdot (i_{\mathrm{OPT}} + b_{\mathrm{OPT}}) - 2 \\ &\leq 3 \cdot R_{\mathrm{OPT}}. \end{aligned}$$

□

5.4 Online Maximum Area Triangulation

While we presented a competitive strategy for the OMRTP we prove in this section that the OMATP does not allow for a competitive algorithm. Recall that we ask for a connected triangulation.

Theorem 15. *There is no competitive algorithm for the Online Maximum Area Triangulation Problem.*

Proof. We construct a polygon in which every online algorithm for the OMATP covers less than $\frac{8}{\sqrt{3}}\varepsilon$ of the area that the optimal offline algorithm OPT covers for any given ε (using ℓ relays).

We start with two relays located on vertices within a distance 1 from each other; let those locations be a and b. First, we construct a rectangle of height 2ε and length 1 above those; see Figure 5.18. On the long side of this rectangle we connect x corridors of width δ and length $2\sqrt{1-\delta^2}$, with $x = \lceil \frac{\ell-2}{4} \rceil$. We choose δ such that $\frac{1}{2} \cdot \delta \cdot (2\sqrt{1-\delta^2} + \mu) \leq \varepsilon$, for some (small) $\mu \geq 0$ (then of course $A = \frac{1}{2} \cdot \delta \cdot \sqrt{1-\delta^2} \leq \varepsilon$ for the area of the triangles in the corridors).

$(x-1)$ of the x corridors have a small wider structure at the end; see Figure 5.18 (the total length of those corridors is $2+\mu$). One of the corridors of width δ and length $2\sqrt{1-\delta^2}$ ends in a polygonal piece with area sufficient for the placement of ℓ regular triangles; see Figure 5.18 for the entire construction.

Thus, for every triangle in the ab-rectangle we have an area of A: $A \leq \frac{1}{2} \cdot 1 \cdot 2\varepsilon = \varepsilon$. Any online algorithm has to choose locations of relays in the corridors at some point, either exploring corridors one after another or placing relays in different corridors. For a single corridor 4 robots need to be placed before it is possible to distinguish the dead ends from the large area.

As we consider an online scenario, the corridor in which an online algorithm ALG will place a 4th relay last will feature the large polygonal piece. Consequently, ALG places 2 relays at a and b, $(x-1) \cdot 4$ relays in the $(x-1)$ corridors ending in a small triangle and $(\ell - 2 - (x-1) \cdot 4) \leq 4$ relays in the corridor with the large polygonal piece. The optimum constructs one triangle in the ab-rectangle, two in the corridor, and the rest will be regular triangles with $A = \frac{\sqrt{3}}{4}$ in the large polygonal piece.

Thus, altogether we have:

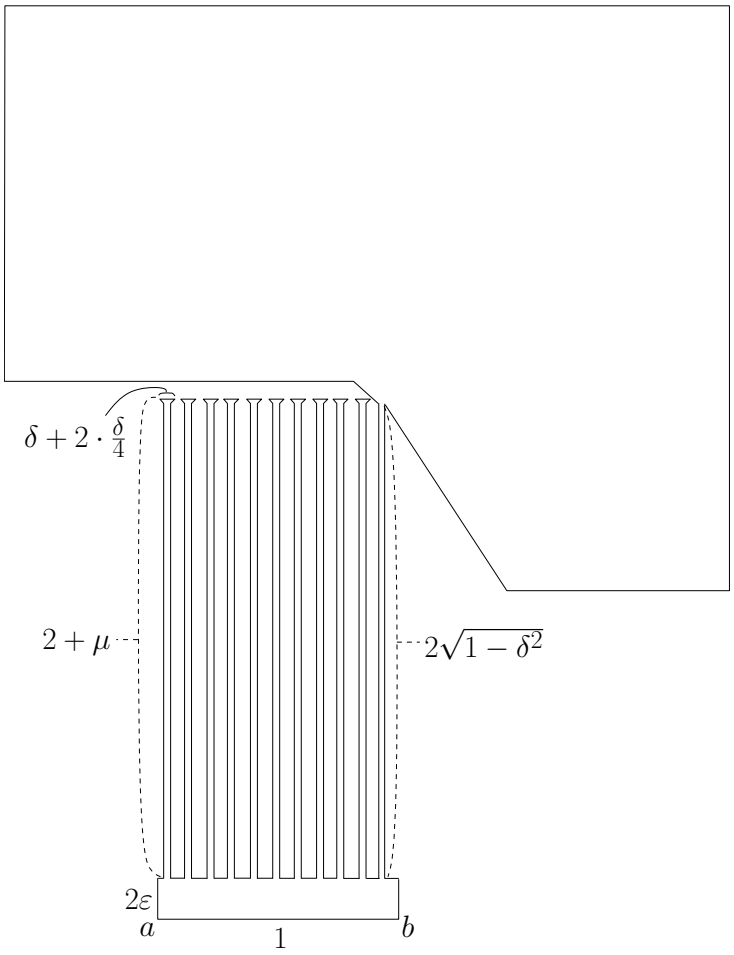


Figure 5.18: An example for the polygon construction with $x = 11$.

$$A_{\text{ALG}} \leq \ell \cdot \varepsilon \tag{5.14}$$

$$A_{\text{OPT}} \geq (\ell - 7) \cdot \frac{\sqrt{3}}{4} \geq \frac{\ell}{2} \cdot \frac{\sqrt{3}}{4} \qquad (\ell \geq 14) \tag{5.15}$$

$$\Rightarrow \quad \frac{A_{\text{ALG}}}{A_{\text{OPT}}} < \frac{\ell \cdot \varepsilon}{\frac{\ell}{2} \cdot \frac{\sqrt{3}}{4}} = \frac{8\varepsilon}{\sqrt{3}} \qquad (\ell \geq 14) \tag{5.16}$$

□

5.5 Conclusion

We studied the problem of monitoring a polygon with a swarm of robots with limited communication range. The robots start the exploration from a given

starting point, decide on locations and construct a triangulated network. The edges in this triangulation must not exceed a length of 1. For the MRTP the entire polygon is to be covered and we ask for the minimum number of relays necessary for this task. For the MATP we are equipped with a certain number of relays and want to maximize the area covered by the established triangulation.

We gave offline and online results for these problems.

For the online MRTP we proved a lower bound of 6/5 on the competitive ratio of any online algorithm and presented a 3-competitive strategy. Obviously, there is still gap between the upper and lower bound on the competitive factor. At this point, the bottleneck in our analysis of our strategy is that we account for boundary, interior and "connection" relays separately. Consequently, there is still room for improvement: Our analysis might be improved or there might be more sophisticated strategies.

We showed that the OMATP does not allow for a strategy with a constant competitive factor. This is due to the requirement of fixing a location during the exploration process. Thus, changing the basic nature of the problem, and allowing for a continued movement (while staying connected) might result in positive results—for the revised problem.

For the offline versions of both problems we gave a NP-hardness proof. Note that for slightly altered problem statements Mitchell (see [FKK+11]) presented polynomial time approximation schemes based on m-guillotine subdivisions. This variant adds two natural assumptions, rooted in the robots' finite size: The region may be assumed to be free of bottlenecks that are too narrow for robots, and we may already have a discrete set of candidate relay locations. On the other hand, if we do not allow for the additional restrictions, the 3-competitive strategy that we presented for the OMRTP of course also gives a 3-approximation algorithm. Consequently, trying to improve this approximation factor for the general MRTP and giving a constant-factor approximation algorithm for the MATP without the additional assumptions could be of interest.

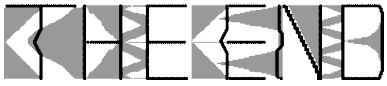
CHAPTER 6

Conclusion

In this thesis we presented results for three different geometric problems using mobile agents with limited capabilities—in the context of exploration, guarding and image analysis. We already gave more detailed summaries and discussed some open problems in Section 3.5, Section 4.6 and Section 5.5.

We considered the problem of extracting certain object attributes from a given smart pixel image in Chapter 3. This differs from a classical camera chip as pixels may perform simple computations, communicate with direct neighbors and react to events. We showed how the desired attributes, the object's size, its center of gravity and its orientation, can be expressed as moments (considering the occupied pixels as a point set in $\mathbb{R}^2$). We used the pixels to mimic mobile agents, messages sent from a pixel to a neighboring pixel. Moreover, we showed how a weight assignment to object pixels and a consumption of these weights by agents, can be performed using simple computations only and allows for the computation of the moments. We presented an algorithm that ensures these computations and proved a linear runtime. The algorithm can handle arbitrarily intertwined objects in the given picture.

In Chapter 4 we presented the Myopic Watchman Problem with Discrete Vision. This problem imposes two additional constraints on the classical watchman route problem: The watchman has a limited range of visibility (scan range) and may perceive the environment at discrete points along its route only. Consequently, the cost of a tour is given as a linear combination of tour length and number of scan points. We prove that the additional constraints result in an NP-hard problem. We presented approximation al-



gorithms for different variants. We achieved a 2.5- and 4-approximation strategy in rectilinear grid polygons for explorers with a unit L_∞ and L_2 scan range, respectively. For general polygons, and L_2 scan range we bound the ratio between scan range r and minimum side length a and presented a $\max(\frac{21}{4}, \frac{\pi r}{a} + \frac{\pi r}{2} + \frac{\pi}{2})$-approximation.

The Minimum Relay Triangulation Problem and the Maximum Area Triangulation Problem are studied in Chapter 5. Both problems are based on the same situation. Relays with a limited communication range should explore a polygon from a starting point in a connected fashion. The relays have to decide on their locations resulting in a triangulated network in the polygon, such that the edge lengths in the triangulation do not exceed the communication range. The first problem asks for the minimum number of relays necessary for complete coverage of the polygon. We showed that this problem is NP-hard. The focus was on the online setting. We gave a lower bound of 6/5 for the competitive factor of any online strategy and presented a 3-competitive strategy. This strategy also provides a 3-approximation for the offline case. The second problem asks for the maximum area that can be covered using a given number of relays. Also this problem was shown to be NP-hard. Again, our focus was on online algorithms. But, we proved that no deterministic algorithm can achieve a constant competitive ratio.

List of Figures

Bibliography

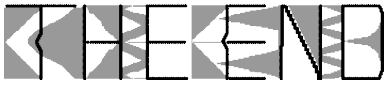
[AAB+06] Helmut Alt, Esther M. Arkin, Hervé Brönnimann, Jeff Erickson, Sándor P. Fekete, Christian Knauer, Jonathan Lenchner, Joseph S. B. Mitchell, and Kim Whittlesey. Minimum-cost coverage of point sets by disks. In *SCG '06: Proceedings of the twenty-second annual symposium on Computational geometry*, pages 449–458, 2006.

[AABV98] Baruch Awerbuch, Yossi Azar, Avrim Blum, and Santosh Vempala. New approximation guarantees for minimum-weight k-trees and prize-collecting salesmen. *SIAM Journal on Computing*, 28:254–262, 1998.

[ABCP96] Baruch Awerbuch, Bonnie Berger, Lenore Cowen, and David Peleg. Fast distributed network decompositions and covers. *Journal of Parallel and Distributed Computing*, 39:105–114, 1996.

[ABD+01] Esther M. Arkin, Michael A. Bender, Erik D. Demaine, Sándor P. Fekete, Joseph S. B. Mitchell, and Saurabh Sethia. Optimal covering tours with turn costs. In *SODA '01: Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete algorithms*, pages 138–147, Philadelphia, PA, USA, 2001. Society for Industrial and Applied Mathematics.

[ACKO88] Larry J. Aupperle, Harold E. Conn, J. Mark Keil, and Joseph O'Rourke. Covering orthogonal polygons with squares. In *Pro-*



ceedings of the 26th Annual Allerton Conference on Communication, Control, and Computing, pages 97–106, October 1988.

[AFM00] Esther M. Arkin, Sándor P. Fekete, and Joseph S. B. Mitchell. Approximation algorithms for lawn mowing and milling. *Computational Geometry: Theory and Applications*, 17(1-2):25–50, 2000.

[AK84] Mikhail J. Atallah and S. Rao Kosaraju. Graph problems on a mesh-connected processor array. *Journal of the ACM*, 31:649–667, June 1984.

[AK00] Franz Aurenhammer and Rolf Klein. Voronoi diagrams. In Jörg-Rüdiger Sack and Jorge Urrutia, editors, *Handbook of Computational Geometry*, pages 201–290. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.

[AKM93] Esther M. Arkin, Samir Khuller, and Joseph S. B. Mitchell. Geometric knapsack problems. *Algorithmica*, 10:399–427, 1993.

[AKS99] Susanne Albers, Klaus Kursawe, and Sven Schuierer. Exploring unknown environments with obstacles. In *SODA '99: Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete algorithms*, pages 842–843, 1999.

[AMN98] Esther M. Arkin, Joseph S. B. Mitchell, and Giri Narasimhan. Resource-constrained geometric network optimization. In *SCG '98: Proceedings of the fourteenth annual symposium on Computational geometry*, pages 307–316, 1998.

[AMP10] Yoav Amit, Joseph S. B. Mitchell, and Eli Packer. Locating guards for visibility coverage of polygons. *International Journal of Computational Geometry & Applications*, 20(5):601–630, 2010.

[Aro98] Sanjeev Arora. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *Journal of the ACM*, 45(5):753–782, September 1998.

[AT81] David Avis and Godfried Toussaint. An efficient algorithm for decomposing a polygon into star-shaped pieces. *Pattern Recognition*, 13:395–398, 1981.

[Awe87] Baruch Awerbuch. Optimal distributed algorithms for minimum weight spanning tree, counting, leader election, and related problems. In *STOC '87: Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, pages 230–240, New York, NY, USA, 1987. ACM.

[BB82] Dana H. Ballard and Christopher M. Brown. *Computer Vision*. Prentice Hall, 1982.

[BCK+07] Avrim Blum, Shuchi Chawla, David R. Karger, Terran Lane, Adam Meyerson, and Maria Minkoff. Approximation algorithms for orienteering and discounted-reward TSP. *SIAM Journal on Computing*, 37:653–670, 2007.

[BDDG82] James E. Boyce, David P. Dobkin, Robert L.(Scot) Drysdale, III, and Leo J. Guibas. Finding extremal polygons. In *STOC '82: Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 282–289, New York, NY, USA, 1982. ACM.

[BE92] Marshall Bern and David Eppstein. Mesh generation and optimal triangulation. In D.-Z. Du and F. K. Hwang, editors, *Computing in Euclidean Geometry*, volume 1 of *Lecture Notes Series on Computing*, pages 23–90. World Scientific, Singapore, 1992.

[BEG94] Marshall Bern, David Eppstein, and John Gilbert. Provably good mesh generation. *Journal of Computer and System Sciences*, 48:384–409, June 1994.

[BEY98] Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, New York, NY, USA, 1998.

[BF01] Christoph Baur and Sándor P. Fekete. Approximation of geometric dispersion problems. *Algorithmica*, 30(3):451–470, 2001.

[BFKS10] Tobias Baumgartner, Sándor P. Fekete, Alexander Kröller, and Christiane Schmidt. Exact solutions and bounds for general art gallery problems. In *Proceedings of the SIAM-ACM Workshop on Algorithm Engineering and Experiments, ALENEX 2010*, pages 11–22. SIAM, 2010.

[BG94] Hervé Brönnimann and Michael T. Goodrich. Almost optimal set covers in finite VC-dimension. In *SCG '94: Proceedings of the 10th annual symposium on Computational geometry*, pages 293–302, 1994.

[BGS01] Amitava Bhattacharya, Subir Kumar Ghosh, and Sudeep Sarkar. Exploring an unknown polygonal environment with bounded visibility. In *International Conference on Computational Science (1)*, volume 2073 of *LNCS*, pages 640–648. Springer, 2001.

[BMKM05] Boaz Ben-Moshe, Matthew J. Katz, and Joseph S. B. Mitchell. A constant-factor approximation algorithm for optimal terrain guarding. In *SODA '05: Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '05, pages 515–524, Philadelphia, PA, USA, 2005. Society for Industrial and Applied Mathematics.

[BMS+08] Jan Brunner, Matúš Mihalák, Subhash Suri, Elias Vicari, and Peter Widmayer. Algorithmic aspects of wireless sensor networks. chapter Simple Robots in Polygonal Environments: A Hierarchy, pages 111–124. Springer-Verlag, Berlin, Heidelberg, 2008.

[CdRdS09] Marcelo C. Couto, Pedro J. de Rezende, and Cid C. de Souza. An exact algorithm for an art gallery problem. Technical Report IC-09-46, Institute of Computing, University of Campinas, November 2009.

[CdSdR08] Marcelo C. Couto, Cid C. de Souza, and Pedro Jussieu de Rezende. Experimental evaluation of an exact algorithm for the orthogonal art gallery problem. In *WEA '08: 7th International Workshop on Experimental Algorithms*, pages 101–113, 2008.

[Cha91] Bernard Chazelle. Triangulating a simple polygon in linear time. *Discrete & Computational Geometry*, 6:485–524, 1991.

[Cha03] Timothy M. Chan. Polynomial-time approximation schemes for packing and piercing fat objects. *Journal of Algorithms*, 46:178–189, February 2003.

[Che89] L. Paul Chew. Guaranteed-quality triangular meshes. Technical Report TR-89-983, Department of Computer Science, Cornell University, Ithaca, NY, April 1989.

[Chv74] Vasek Chvátal. A combinatorial theorem in plane geometry. *Journal of Combinatorial Theory*, ((B)18):39–41, 1974.

[CJN93] Svante Carlsson, Håkan Jonsson, and Bengt J. Nilsson. Finding the shortest watchman route in a simple polygon. In *Algorithms and Computation, 4th International Symposium, ISAAC '93, Proceedings*, pages 58–67, 1993.

[CN86] Wei-Pang Chin and Simeon Ntafos. Optimum watchman routes. In *SCG '86: Proceedings of the second annual symposium on Computational geometry*, pages 24–33, New York, NY, USA, 1986. ACM.

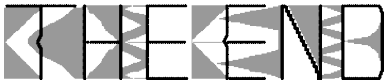
[CN91] Wei-Pang Chin and Simeon C. Ntafos. Shortest watchman routes in simple polygons. *Discrete & Computational Geometry*, 6:9–31, 1991.

[CNN93] Svante Carlsson, Bengt J. Nilsson, and Simeon C. Ntafos. Optimum guard covers and m-watchmen routes for restricted polygons. *International Journal on Computational Geometry and Applications*, 3(1):85–105, 1993.

[CR88] Joseph C. Culberson and Robert A. Reckhow. Covering polygons is hard. In *29th Annual Symposium on Foundations of Computer Science, FOCS '88*, pages 601–611, 1988.

[CSR07] Marcelo C. Couto, Cid C. de Souza, and Pedro J. de Rezende. An exact and efficient algorithm for the orthogonal art gallery problem. In *SIBGRAPI '07: Proceedings of the XX Brazilian Symposium on Computer Graphics and Image Processing*, pages 87–94, Washington, DC, USA, 2007. IEEE Computer Society.

[CV05] Kenneth L. Clarkson and Kasturi Varadarajan. Improved approximation algorithms for geometric set cover. In *SCG '05: Proceedings of the 21st annual symposium on Computational geometry*, pages 135–141, New York, NY, USA, 2005. ACM.



[dBCvKO08] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer, 2008.

[DKP98] Xiaotie Deng, Tiko Kameda, and Christos H. Papadimitriou. How to learn an unknown environment I: The rectilinear case. *Journal of the ACM*, 45(2):215–245, 1998.

[DM03] Adrian Dumitrescu and Joseph S. B. Mitchell. Approximation algorithms for TSP with neighborhoods in the plane. *Journal of Algorithms*, 48(1):135–159, 2003.

[EFG+08] Alon Efrat, Sándor P. Fekete, Poornananda R. Gaddehosur, Joseph S. Mitchell, Valentin Polishchuk, and Jukka Suomela. Improved approximation algorithms for relay placement. In *Proceedings of the 16th Annual European Symposium on Algorithms*, pages 356–367. Springer, 2008.

[EHP06] Alon Efrat and Sariel Har-Peled. Guarding galleries and terrains. *Information Processing Letters*, 100(6):238 –245, 2006.

[EKM+11] Khaled Elbassioni, Erik Krohn, Domagoj Matijević, Julián Mestre, and Domagoj Ševerdija. Improved approximations for guarding 1.5-dimensional terrains. *Algorithmica*, 60:451–463, June 2011.

[Elk06] Michael Elkin. A faster distributed protocol for constructing a minimum spanning tree. *Journal of Computer and System Sciences*, 72(8):1282 – 1308, 2006.

[Epp92] David Eppstein. Approximating the minimum weight triangulation. In *SODA '92: Proceedings of the third annual ACM-SIAM symposium on Discrete algorithms*, pages 48–57, 1992.

[Epp01] David Eppstein. Global optimization of mesh quality. http://www.ics.uci.edu/ eppstein/pubs/Epp-IMR-01.pdf, 2001. Tutorial at 10th International Meshing Roundtable.

[ESW01] Stephan Eidenbenz, Christoph Stamm, and Peter Widmayer. Inapproximability results for guarding polygons and terrains. *Algorithmica*, 31(1):79–113, 2001.

[FFK+09] Sándor P. Fekete, Dietmar Fey, Marcus Komann, Alexander Kröller, Marc Reichenbach, and Christiane Schmidt. Distributed vision with smart pixels. In *SCG '09: Proceedings of the 25th annual symposium on Computational geometry*, pages 257–266, New York, NY, USA, 2009. ACM.

[FHL+04] Dietmar Fey, Lutz Hoppe, Andreas Loos, Michael Förtsch, and Horst Zimmermann. Parallel optical interconnects with mixed-signal OEIC and fibre arrays for high-speed communication. In *Proceedings of SPIE*, volume 5453 of *Micro-Optics, VCSELs and Photonic Interconnects*, Strasbourg, France, 2004. Photonics Europe.

[Fis78] Steve Fisk. A short proof of Chvátal's watchman theorem. *Journal of Combinatorial Theory*, (B 24):374–375, 1978.

[FK84] Deborah S. Franzblau and Daniel J. Kleitman. An algorithm for constructing regions with rectangles: Independence and minimum generating sets for collections of intervals. In *STOC '84: Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 167–174, New York, NY, USA, 1984. ACM.

[FKK+11] Sándor P. Fekete, Tom Kamphans, Alexander Kröller, Joseph S. B. Mitchell, and Christiane Schmidt. Exploring and triangulating a region by a swarm of robots, 2011. Submitted for publication.

[FLMM04] Andras Ferencz, Erik G. Learned-Miller, and Jitendra Malik. Learning hyper-features for visual identification. In *Neural Information Processing Systems*, 2004.

[FMS09] Sándor P. Fekete, Joseph S. B. Mitchell, and Christiane Schmidt. Minimum covering with travel cost. In *Proceedings of the 20th International Symposium on Algorithms and Computation*, volume 5878 of *LNCS*, pages 393–402. Springer, 2009.

[FMS10] Sándor Fekete, Joseph Mitchell, and Christiane Schmidt. Minimum covering with travel cost. *Journal of Combinatorial Optimization*, pages 1–20, 2010.

[FS10] Sándor P. Fekete and Christiane Schmidt. Polygon exploration with time-discrete vision. *Computational Geometry: Theory and Applications*, 43(2):148 – 168, 2010.

[FW98] Amos Fiat and Gerhard J. Woeginger, editors. *Online Algorithms, The State of the Art (the book grow out of a Dagstuhl Seminar, June 1996)*, volume 1442 of *Lecture Notes in Computer Science*. Springer, 1998.

[GBBS08] Subir Kumar Ghosh, Joel W. Burdick, Amitava Bhattacharya, and Sudeep Sarkar. Online algorithms with discrete visibility - exploring unknown polygonal environments. *Robotics Automation Magazine, IEEE*, 15(2):67 –76, 2008.

[Gho10] Subir Kumar Ghosh. Approximation algorithms for art gallery problems in polygons and terrains. In Md. Rahman and Satoshi Fujita, editors, *WALCOM: Algorithms and Computation*, volume 5942 of *Lecture Notes in Computer Science*, pages 21–34. Springer Berlin / Heidelberg, 2010.

[GHS83] Robert G. Gallager, Pierre A. Humblet, and Philip M. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Transactions on Programming Languages and Systems*, 5:66–77, January 1983.

[GJ90] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[GK10] Subir Kumar Ghosh and Rolf Klein. Online algorithms for searching and exploration in the plane. *Computer Science Review*, 4(4):189 – 201, 2010.

[GKKV09] Matt Gibson, Gaurav Kanade, Erik Krohn, and Kasturi Varadarajan. An approximation scheme for terrain guarding. In *Proceedings of the 12th International Workshop and 13th International Workshop on Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, APPROX '09 / RANDOM '09, pages 140–148, Berlin, Heidelberg, 2009. Springer-Verlag.

[GR03] Yoav Gabriely and Elon Rimon. Competitive on-line coverage of grid environments by a mobile robot. *Computational Geometry: Theory and Applications*, 24(3):197–224, 2003.

[GW08] Rafael C. Gonzalez and Richard E. Woods. *Digital image processing*. Prentice-Hall, third edition, 2008.

[HAB+02] Tien-Ruey Hsiang, Esther M. Arkin, Michael A. Bender, Sándor P. Fekete, and Joseph S. B. Mitchell. Algorithms for rapidly dispersing robot swarms in unknown environments. In *Proceedings of the 5th Workshop Algorithmic on the Algorithmic Foundations of Robotics*, pages 77–94, 2002.

[HAB+03] Tien-Ruey Hsiang, Esther M. Arkin, Michael A. Bender, Sándor P. Fekete, and Joseph S. B. Mitchell. Online dispersion algorithms for swarms of robots. In *SCG '03: Proceedings of the nineteenth annual symposium on Computational geometry*, pages 382–383, 2003. Video.

[HIKK97] Frank Hoffmann, Christian Icking, Rolf Klein, and Klaus Kriegel. A competitive strategy for learning a polygon. In *SODA '97: Proceedings of the Eigth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 166–174, 1997.

[HIKK01] Frank Hoffmann, Christian Icking, Rolf Klein, and Klaus Kriegel. The polygon exploration problem I: A competitive strategy. *SIAM Journal on Computing*, 31:577–600, 2001.

[HM85] Dorit S. Hochbaum and Wolfgang Maass. Approximation schemes for covering and packing problems in image processing and VLSI. *Journal of the ACM*, 32(1):130–136, 1985.

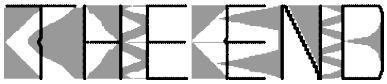
[HNP06] Mikael Hammar, Bengt J. Nilsson, and Mia Persson. Competitive exploration of rectilinear polygons. *Theoretical Computer Science*, 354(3):367 – 378, 2006. Foundations of Computation Theory (FCT 2003).

[HNS02] Mikael Hammar, Bengt J. Nilsson, and Sven Schuierer. Improved exploration of rectilinear polygons. *Nordic Journal of Computing*, 9:32–53, March 2002.

[Hoc97] Dorit S. Hochbaum, editor. *Approximation algorithms for NP-hard problems*. PWS Publishing Co., Boston, MA, USA, 1997.

[HW86] David Haussler and Emo Welzl. Epsilon-nets and simplex range queries. In *SoCG '86: Proceedings of the Second Annual Symposium on Computational Geometry*, pages 61–71, 1986.

[IKKL05] Christian Icking, Tom Kamphans, Rolf Klein, and Elmar Langetepe. Exploring simple grid polygons. In *11th International Computing and Combinatorics Conference*, volume 3595 of *LNCS*, pages 524–533. Springer, 2005.



[IPS82] Alon Itai, Christos H. Papadimitriou, and Jayme Luiz Szwarcfiter. Hamilton paths in grid graphs. *SIAM Journal on Computing*, 11(4):676–686, 1982.

[JM08] Klaus Jansen and Marian Margraf. *Approximative Algorithmen und Nichtapproximierbarkeit*. de Gruyter, 2008.

[Jol02] Ian T. Jolliffe. *Principal Component Analysis*. Springer, second edition, 2002.

[Kei86] J. Mark Keil. Minimally covering a horizontally convex orthogonal polygon. In *SCG '86: Proceedings of the second annual symposium on Computational geometry*, pages 43–51, 1986.

[Ker39] Richard Kershner. The number of circles covering a set. *American Journal of Mathematics*, 61:665–67, 1939.

[Kin06] James King. A 4-approximation algorithm for guarding 1.5-dimensional terrains. In José Correa, Alejandro Hevia, and Marcos Kiwi, editors, *LATIN 2006: Theoretical Informatics*, volume 3887 of *Lecture Notes in Computer Science*, pages 629–640. Springer Berlin / Heidelberg, 2006.

[KKK83] Jeff Kahn, Maria Klawe, and Daniel Kleitman. Traditional art galleries require fewer watchmen. *SIAM Journal on Algebraic and Discrete Methods*, 4(2):194–206, 1983.

[KKS+08] Marcus Komann, Alexander Kröller, Christiane Schmidt, Dietmar Fey, and Sándor P. Fekete. Emergent algorithms for centroid and orientation detection in high-performance embedded cameras. In *CF '08: Proceedings of the 2008 conference on Computing frontiers*, pages 221–230, New York, NY, USA, 2008. ACM.

[Kle94] Jon M. Kleinberg. On-line search in a simple polygon. In *SODA '94: Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 8–15, 1994.

[KR92] Donald E. Knuth and Arvind Raghunathan. The problem of compatible representatives. *SIAM Journal of Discrete Math.*, 5(3):422–427, 1992.

[Krö10] Alexander Kröller. Personal communication, 2010.

[Lau99] Aldo Laurentini. Guarding the walls of an art gallery. *The Visual Computer*, 15(6):265–278, 1999.

[LL86] Der-Tsai Lee and Arthur K. Lin. Computational complexity of art gallery problems. *IEEE Transactions on Information Theory*, 32(2):276–282, 1986.

[LLRS85] Eugene L. Lawler, Jan Karel Lenstra, Alexander H. G. Rinnooy Kan, and David B. Shmoys, editors. *The Traveling Salesman Problem*. John Wiley & Sons, New York, NY, 1985.

[Lyn96] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996.

[Mit99] Joseph S. B. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems. *SIAM Journal on Computing*, 28:1298–1309, 1999.

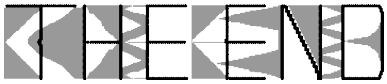
[Mit00] Joseph S. B. Mitchell. Geometric shortest paths and network optimization. In Jörg-Rüdiger Sack and Jorge Urrutia, editors, *Handbook of Computational Geometry*, pages 633–701. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.

[Mit07] Joseph S. B. Mitchell. A PTAS for TSP with neighborhoods among fat regions in the plane. In *SODA '07: Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 11–18, 2007.

[MR08] Wolfgang Mulzer and Günter Rote. Minimum-weight triangulation is NP-hard. *Journal of the ACM*, 55:11:1–11:29, May 2008.

[MR09] Nabil Hassan Mustafa and Saurabh Ray. PTAS for geometric hitting set problems via local search. In *SCG '09: Proceedings of the 25th annual symposium on Computational geometry*, pages 17–22, New York, NY, USA, 2009. ACM.

[MS85] Russ Miller and Quentin F. Stout. Geometric algorithms for digitized pictures on a mesh-connected computer. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-7:216–228, 1985.



[MS88] Russ Miller and Quentin F. Stout. Efficient parallel convex hull algorithms. *IEEE Transactions on Computers*, C-37(12):1605–1618, 1988.

[MS07] James McLurkin and Jennifer Smith. Distributed algorithms for dispersion in indoor environments using a swarm of autonomous mobile robots. In Rachid Alami, Raja Chatila, and Hajime Asama, editors, *Distributed Autonomous Robotic Systems 6*. Springer Japan, 2007.

[Nil01] Bengt J. Nilsson. Approximating a shortest watchman route. *Fundamenta Informaticae*, 45:235–281, January 2001.

[Nil05] Bengt J. Nilsson. Approximate guarding of monotone and rectilinear polygons. In Luís Caires, Giuseppe Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *Automata, Languages and Programming*, volume 3580 of *Lecture Notes in Computer Science*, pages 1362–1373. Springer Berlin / Heidelberg, 2005.

[NS80] David Nassimi and Sartaj Sahni. Finding connected components and connected ones on a mesh-connected parallel computer. *SIAM Journal on Computing (SICOMP)*, 9:744–757, 1980.

[O'R87] Joseph O'Rourke. *Art Gallery Theorems and Algorithms*. International Series of Monographs on Computer Science. Oxford University Press, New York, NY, 1987.

[OS83] Joseph O'Rourke and Kenneth Supowit. Some NP-hard polygon decomposition problems. *Information Theory, IEEE Transactions on*, 29(2):181–190, Mar 1983.

[Pap77] Christos H. Papadimitriou. The Euclidean traveling salesman problem is NP-complete. *Theoretical Computer Science*, 4:237–244, 1977.

[Pap94] Christos M. Papadimitriou. *Computational complexity*. Addison-Wesley, Reading, Massachusetts, 1994.

[Pel00] David Peleg. *Distributed computing: a locality-sensitive approach*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[RT86] Pierre Rosenstiehl and Robert Endre Tarjan. Rectilinear planar layouts and bipolar orientations of planar graphs. *Discrete & Computational Geometry*, 1:343–353, 1986.

[Sch89] Otfried Schwarzkopf. Parallel computation of discrete Voronoi diagrams. In *Proceedings of the 6th Symposium of Theoretical Aspects of Computer Science*, volume 349, pages 193–204. Springer-Verlag, 1989.

[Sch03] Alexander Schrijver. *Combinatorial Optimization - Polyhedra and Efficiency*. Springer, 2003.

[SH95] Dietmar Schuchardt and Hans-Dietrich Hecker. Two NP-hard art-gallery problems for ortho-polygons. *Mathematical Logic Quarterly*, 41:261–267, 1995.

[She92] Thomas C. Shermer. Recent results in art galleries. In *Proceedings of the IEEE*, volume 80, pages 1384–1399, 1992.

[Shl05] Jonathon Shlens. A tutorial on principal component analysis, 2005. http://www.snl.salk.edu/ shlens/pca.pdf.

[SVW08] Subhash Suri, Elias Vicari, and Peter Widmayer. Simple robots with minimal sensing: From local visibility to global geometry. *International Journal of Robotics Research*, 27:1055–1067, September 2008.

[TF80] Steven L. Tanimoto and Robert J. Fowler. Covering image subsets with patches. *NATO Conference Series, (Series) 4: Marine Sciences*, 2:835–839, 1980.

[TH93] Xuehou Tan and Tomio Hirata. Constructing shortest watchman routes by divide-and-conquer. In *Algorithms and Computation, 4th International Symposium, ISAAC '93, Proceedings*, pages 68–77, 1993.

[THI91] Xuehou Tan, Tomio Hirata, and Yasuyoshi Inagaki. An incremental algorithm for constructing shortest watchman routes. In Wen-Lian Hsu and Richard C. T. Lee, editors, *ISA*, volume 557 of *Lecture Notes in Computer Science*, pages 163–175. Springer, 1991.

[THI99] Xuehou Tan, Tomio Hirata, and Yasuyoshi Inagaki. Corrigendum to "an incremental algorithm for constructing shortest watchman routes". *International Journal of Computational Geometry and Applications*, 9(3):319–323, 1999.

[TK77] Clark D. Thompson and H. T. Kung. Sorting on a mesh-connected parallel computer. *Commununications of the ACM*, 20:263–271, 1977.

[Tót49] László Fejes Tóth. Über dichteste Kreislagerung und dünnste Kreisüberdeckung. *Commentarii Mathematici Helvetici*, 23(1):342–349, 1949.

[Urr00] Jorge Urrutia. Art gallery and illumination problems. In Jörg-Rüdiger Sack and Jorge Urrutia, editors, *Handbook on Computational Geometry*, pages 973–1026. Elsevier Science Publishers, 2000.

[Vaz01] Vijay V. Vazirani. *Approximation algorithms*. Springer, 2001.

[VS80] Frances L. Van Scoy. The parallel recognition of classes of graphs. *Computers, IEEE Transactions on*, C-29(7):563 –570, 1980.

[WK07] Chris Worman and J. Mark Keil. Polygon decomposition and the orthogonal art gallery problem. *International Journal of Computational Geometry and Applications*, 17(2):105–138, 2007.

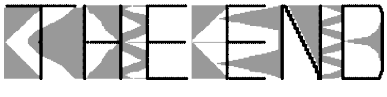
[WKG07a] Pengpeng Wang, Ramesh Krishnamurti, and Kamal Gupta. Generalized watchman route problem with discrete view cost. In *Proceedings of the 19th Canadian Conference on Computational Geometry (CCCG2007)*, pages 241–244, 2007.

[WKG07b] Pengpeng Wang, Ramesh Krishnamurti, and Kamal Gupta. View planning problem with combined view and traveling cost. In *2007 IEEE International Conference on Robotics and Automation, ICRA'07*, pages 711–716, 2007.

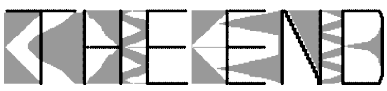
[WLB00] Israel A. Wagner, Michael Lindenbaum, and Alfred M. Bruckstein. MAC vs. PC: Determinism and randomness as complementary approaches to robotic exploration of continuous unknown domains. *ROBRES: The International Journal of Robotics Research*, 19(1):12–31, 2000.

[WRR03] Michael E. Wall, Andreas Rechtsteiner, and Luis M. Rocha. Singular value decomposition and principal component analysis. In Daniel P. Berrar, Werner Dubitzky, and Martin Granzow, editors, *A Practical Approach to Microarray Data Analysis*, pages 91–109. Springer, 2003.

[WS90] San-Yuan Wu and Sartay Sahni. Covering rectilinear polygons by rectangles. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 9(4):377 –388, April 1990.



Index



Curriculum Vitae

Christiane Schmidt
Stobenstr. 12
38100 Braunschweig
Phone (office): (05 31) 3 91 31 14
E-mail: c.schmidt@tu-bs.de

Personal Data

Date of birth:	27.06.1981
Place of birth:	Wolfsburg
Nationality:	German

Education

08/1987–06/1991	Primary school (Hellwinkelschule), Wolfsburg.
08/1991–06/1993	5th and 6th grade (Hellwinkelschule), Wolfsburg.
08/1993–06/2000	Grammar school (Gymnasium Fallersleben), Wolfsburg. Overall grade: 1.1 .

Academic Studies

10/2000–02/2006	Studies of *Financial and Business Mathematics* at Braunschweig University of Technology. Major in mathematical optimization.
	Diploma thesis *Polygon Exploration with Scan Costs*.
03/2006	Diploma in "Financial and Business Mathematics". Overall grade: 1.1 .

Work Experience

05/2006–09/2007	Research assistant at the Institute of Mathematical Optimization, Braunschweig University of Technology.
since 10/2007	Research assistant at the IBR Algorithms Group, Braunschweig University of Technology.