On some Geometric Search Problems

Algorithms, Complexity, Computability

David J. F. Kübel
aus Bayreuth

Mathematisch-Naturwissenschaftliche Fakultät
Rheinische Friedrich-Wilhelms-Universität Bonn

Dissertation zur Erlangung des Doktorgrades
Doctor rerum naturalium

Bonn, 2020

1. Gutachter: PD Dr. Elmar Langetepe
2. Gutachter: Prof. Dr. Heiko Röglin

Prüfungsdatum: 15.09.2020
Erscheinungsjahr: 2020

Angefertigt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der
Rheinischen Friedrich-Wilhelms-Universität Bonn

Abstract

This thesis considers four search problems which are related to or emerge from a geometric context. Each search problem is examined in a separate chapter; these four chapters are framed by an introduction and a conclusion which discuss the content at a meta-level. Chapter 1 introduces fundamental concepts and formal notation. Additionally, references to related articles and textbooks for further reading are provided.

Chapter 2 examines the search for a single target point t hiding in Euclidean space. The chapter considers a new query model whereby query points must be placed to locate the target. The feedback of the queries is an ordering of the query points by distance to t. With regard to the feedback, two variants of the query model are distinguished: Either query points have to be placed all at once or one by one depending on the response received on the previous queries. For both variants, we develop lower and upper bounds on the precision that can be achieved.

Chapter 3 examines the search for a boundary point of a simply connected region R in the Euclidean plane. Starting from an unknown position $s \in R$, the searcher follows a trajectory until a boundary point is reached. A competitive analysis is employed to prove that a specific spiral strategy achieves a reasonable competitive factor if R has a non-empty kernel; the competitive factor is even optimal if s lies within the kernel of R. The basis for the competitive analysis is a new measure of intrinsic complexity for instances of geometric search problems. In principle, the lower bound technique can also be applied to prove optimality for other geometric search problems; for many of them, optimality is a long-standing open problem.

Chapter 4 examines the search for a rounding of an edge-weighted graph which meets certain constraints. Given an undirected graph with edge-weight function ω, (how efficient) is it possible to find an integer-valued weight function $\widetilde{\omega}$ such that the absolute change in weight of each shortest path is less than ε? What happens if we add the restriction that a shortest path w. r. t. ω should remain shortest w. r. t. $\widetilde{\omega}$? We further impose the new restriction that a shortest path w. r. t. $\widetilde{\omega}$ must already be shortest w. r. t. ω and settle the complexity status of all three versions of this problem: By reduction from 3-SAT, even the simplest form is NP-hard for general graphs and small values of ε. However, if the graph is a tree with n vertices, an algorithm is given to compute a solution in $O(n^2)$ time.

Chapter 5 examines the search for specific configurations in a new, two-dimensional, discrete fire-expansion model. For example, given two cells s and t where cell s is initially set on fire, will cell t eventually ignite? We prove this question to be undecidable, i. e., no general algorithm exists that can solve every instance of this decision problem within any finite number of steps. In fact, we prove by reduction from two-register machines that the fire-expansion model is Turing-universal, i. e., it is capable of carrying out the computations of arbitrary Turing machines.

Acknowledgements

This thesis would not have been accomplished without the help of many people. It is about time to gratefully acknowledge their advice and support over the years.

To begin with, I would like to thank Elmar Langetepe, Heiko Röglin, Christian Knauer and Andreas Hense for serving on the doctoral committee. Many thanks for your time and expertise.

Sincere thanks go to Rolf Klein for welcoming me into his research group at Department I at the Institute of Computer Science, University of Bonn. This gave me the opportunity to learn a great deal about Theoretical Computer Science. He always encouraged participation at workshops and conferences, initiated discussions about research projects and contributed with major advances.

Moreover, I would like to thank my colleagues from Department I and V. They provided an inspiring and motivating environment for my PhD studies and the joint work on research questions. In particular, I wish to thank Barbara and Carsten with whom I did not only share the office, but also daily struggles. I am also grateful to Heiko Röglin for welcoming me at Department V after Rolf Klein's retirement in 2019.

Special thanks go to the co-authors of papers presented in this thesis. These are Herman Haverkort, Rolf Klein, Sang-Sub Kim, Elmar Langetepe, Jörg-Rüdiger Sack and Barbara Schwarzwald. They provided inspiration for most of the research questions, contributed key ideas, and crucially advanced the work on previous publications as indicated in Chapter 1.

Above all, my thanks go to my mentor Elmar Langetepe for leading me through my PhD studies. He supported and challenged me with interesting research questions from the beginning. While giving me room to pursue my own research ideas and interests, he guided and helped me to advance my skills. Certainly, many results presented in this thesis are due to his support. He encouraged collaborations and supported participation at workshops and conferences. He has always been available to work on open problems and to discuss questions concerning research or teaching. His honest feedback with regard to talks, drafts of papers, or even of this thesis, was invaluable. Thank you, Elmar!

I am also grateful to the University of Bonn and its Graduate Centre for offering advanced training with the doctorate program. During my time as a PhD student, these courses have helped me to develop many additional skills.

Finally, I wish to thank my family for their support and for the time they had to go without me. Special thanks go to my beloved wife Julia, for her endurance over the past years and her extensive support in the home office during Corona time.

Last but not least, I would like to thank those supporters who are not mentioned here by name. This includes unknown researchers who thoroughly reviewed (preliminary forms of) papers. Many of them provided valuable feedback, thought-provoking impulses and unseen references. This also includes developers who contributed to build the free and open-source software that I have been using. Especially the elaborate typesetting system LaTeX (with its numerous packages) and the CUED template which facilitated the work on this thesis tremendously.

Table of Contents

Frequently Used Notations

Roman Symbols

b	a box in the firefighting reduction, usually with coordinates (b_1, b_2)
c	a constant in $\mathbb{N}$; a cell in the cell-graph
d	Euclidean metric
f, g	usually functions
f_M	(partial) function computed by Turing machine M
$G = (V, E)$	a graph where V denotes the set of vertices and E the set of edges
$\mathscr{L}$, $\mathscr{L}_1/\mathscr{L}_2$	a set of lists, the first/second layer of cells
$2M$	a specific two-register machine
M	a specific Turing machine
p, q	usually a (query) point
R, ∂R	region in the Euclidean space and its boundary
s	start (vertex/point/cell) of a path or fire
t	end or target (vertex/point/cell) of a path or fire; a discrete moment in time
u, v	usually vertices in a graph
$y_i, \overline{y}_i$	Boolean variable and its negation

Greek Symbols

α_x	Length of the circular arc segment of radius x around s of an escape path
δ	transition function of a Turing machine, cellular automaton, or artificial neural network
ε	a (small) positive value representing an error term or radius
Γ	alphabet; spiral curve in the Euclidean plane
γ, $\gamma(t)$	an escape path for some region

ω / $\widetilde{\omega}$	real-valued/integer-valued edge-weight function of a graph
π	path or walk in a graph or the Euclidean plane
π_s, $\pi_s(x)$	certificate path, certificate path for x and s
Σ	alphabet (of a Turing machine)
σ, σ_i	string (sequence of characters) over an alphabet, i-th character the string
$\xi(\mathscr{L})$	maximum-traversal-cost of a set of lists
ζ, $\zeta(t)$	an escape strategy for some region

Subscripts

i, j	subscript indices used to enumerate elements
$\mathtt{l}$, $\mathtt{r}$, $\mathtt{t}$, $\mathtt{b}$	subscript indices indicating the left/right/top/bottom direction

Other Symbols

$B_\varepsilon[p]$	closed n-ball of radius ε and center p
$B_\varepsilon(p)$	open n-ball of radius ε and center p
$\mathrm{bs}(p,q)$	bisector (bisecting curve) of two points p, q
$(\mathbb{R}^n, d)$	n-dimensional Euclidean space

Acronyms / Abbreviations

BFS	breadth-first search
CA	cellular automaton
CA	random-access machine
CNF	conjunctive normal form (of a Boolean formula)
DAG	directed acyclic graph
DFS	depth-first search
DNF	disjunctive normal form (of a Boolean formula)
GoL	(Conway's) Game of Life
ILP	integer linear program
ker	kernel of a (polygonal) region in the Euclidean plane
lcm	least common multiple
NN	artificial neural network
SAT	satisfiability problem of Boolean logic
TM	Turing machine

Chapter 1

Introduction

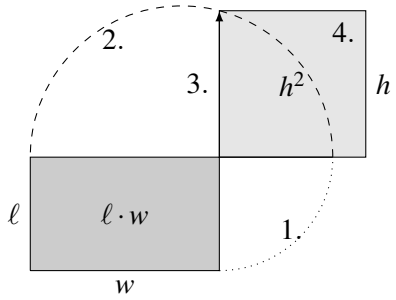
For centuries, people intriguingly studied *algorithms* as systematic descriptions of procedures without a formal definition of the term *algorithm*. To take two examples, consider the following tasks, an arithmetic and a geometric one, cf. Figure 1.1: Given a pair of integer numbers, how can their sum, difference, or product be computed? Or, given a rectangle in the plane, how can one construct a square with the same area as the rectangle using only compass and straight edge? In both examples, a solution to the specific instance can be obtained by carrying out basic operations following a certain procedure, intuitively referred to as an *algorithm*.[1] However, it was not until the 1930s that the term was properly defined by Church, Gödel, Kleene, Turing and others. Their theoretical models of computation paved the way for the invention of modern computers and laid the foundation for modern Computer Science. For this reason, this decade can be considered as the genesis of Theoretical Computer Science, a discipline dedicated to providing a mathematical foundation for past, current and future developments in Computer Science.

A major mission of Theoretical Computer Science is to provide sound mathematical explanations for (computational) questions which arise, e. g., from computer experiments or observations in practice. Reconsider the two initial examples. As far as the *impossibility* of certain classical geometric problems is concerned, Galois theory, a field of Mathematics, answers the question as to whether a Euclidean constructions (algorithm) exists for a given task.[2] However, when it comes to the *efficiency* of computational tasks, no matter whether of arithmetic or geometric nature, it seems fair to say that these are questions primarily examined in (Theoretical) Computer Science. Finding answers to computational questions involves the following two fundamental steps: In a first step, a problem or hypothesis is formally stated; in the second step, it is analysed mathematically. In both steps, an underlying model serves as a cohesive framework interlinking reality and theory; it further provides the basis for a mathematical treatment. The choice of the model must be done carefully: On the one hand, the model needs to be simple enough to abstract from reality and to make the hypothesis mathematically tractable. On the other hand, the model needs to be complex enough that the hypothesis remains meaningful and its statement still accounts for the original context.

By examining four problems in new models, this thesis contributes the following results. The chapters 2 to 5 discuss search problems which are related to or emerge from a geometric context. Each problem is analysed in a new model or context which, compared to previously considered models, adds some power so that surprising and non-trivial results can be proven. This does not only add a certain twist to the problems, it leads to a better understanding of the problems and helps to provide

[1] For a list of characteristics that such an intuitive algorithm should exhibit, see, e. g., [14, p. 3].

[2] For a historical perspective on the notion of algorithms in the field of geometry, we refer to [87, pp. 1 ff.]



(i) Squaring a rectangle with compass and straight edge, i. e., $\ell \cdot w = h^2$.

				2	6	4
$\times$				5	2	1
				2	6	4
$+$			5	2	8	-
$+$	1	3	2	0	-	-
$=$	1	3	7	5	4	4

(ii) The grade-school algorithm for the multiplication applied to 264 and 521.

Figure 1.1: Two different examples of algorithmic procedures. (i) Given a rectangle, one can construct a square of the same area using only ruler and compass by following these four steps. (ii) The grade-school algorithm reduces the initial task to multiplications with a single digit and additions of intermediate results.

mathematical explanations for related problems. The four problems of this thesis can be distinguished via two dimensions: The type of search space (either continuous or discrete) and the number of targets (either a fixed, single target, one arbitrary target from many, or multiple targets at once).

Chapter 2 examines the search for a *single*, stationary target point t in a *continuous* search space. The chapter considers a new query model: Query points are placed in the search space to locate the target; the feedback of the queries is an ordering of the points by distance to t. With respect to the feedback, two variants of the model are distinguished: Either the query points are placed all at once, or points are placed one by one depending on the responses received on the previous queries. Note that the feedback always reveals information about t's location with respect to all previously placed query points; hence, the new query model appears to be stronger than models where feedback only gives information about the most recently issued query, such as, e. g., in a binary search. This raises the question as to whether one can benefit from this additional information. As a matter of fact, such benefits are possible but limited; we prove lower and upper bounds on the precision that can be achieved. These results provide evidence for the observation that binary searching is an efficient approach in many settings.

Chapter 3 also considers a search problem in a *continuous* search space; however, the objective is to find *one out of many* target points. More precisely, located at a starting point s in an unknown, simply connected region R in the Euclidean plane, one must reach ∂R, the boundary of R, within a reasonable amount of time. Time is the limiting cost in this search problem and is measured in terms of the length of the path one follows to reach ∂R. When comparing the cost of a strategy to the shortest path from s to ∂R, it seems impossible to prove the optimality of certain strategies. As an alternative to this standard cost model, we introduce a new cost model for geometric search problems in the plane and discuss the performance of search strategies with respect to this measure. Generally speaking, this cost model opens up an alternative way to prove optimality of geometric search strategies. We apply this technique to prove the optimality of spiral searching strategies for the initial problem and a similar one. The results provide evidence for the observation that spiral searching is highly efficient for many geometric search problems in the Euclidean plane.

In contrast to previous chapters, Chapter 4 examines the search for a solution to the following *discrete* problem where *multiple* constraints must be met: Given an undirected graph with edge-weight

function ω; (how efficient) is it possible to find an integer-valued weight function $\widetilde{\omega}$ such that the absolute change in weight of each shortest path is less than ε? What happens if we add the restriction that a shortest path w. r. t. ω should remain shortest w. r. t. $\widetilde{\omega}$? We further impose the new restriction that a shortest path w. r. t. $\widetilde{\omega}$ must already be shortest w. r. t. ω. The flexibility in this rounding model is given by the error threshold ε, which immediately raises the question whether the difficulty only depends on ε. In fact, such a rounding always exists for rational-valued ω if the error threshold ε is not part of the input and can be chosen arbitrarily large; yet, it is not clear whether this also holds for every real-valued ω. For small, fixed values of ε, we settle the complexity status for all three versions of this problem: By reduction from 3-SAT, even the simplest form of this problem is NP-hard for general graphs. However, if the graph is a tree with n vertices, an algorithm is given to compute a solution in $O(n^2)$ time. These results fit to two general observations in Theoretical Computer Science: First, for many problems[3] better runtime bounds exist for integer-valued than for real-valued input; thus, it seems unlikely that real-valued input can be efficiently transformed into integer-valued input without loss of information in our rounding model. Second, many NP-hard problems on general graphs are easy to solve on restricted graph classes, e. g., vertex cover or dominating set.

Finally, Chapter 5 is motivated by the following search problem for a *single*, specific configuration in a new, two-dimensional, *discrete* fire-expansion model: Given two cells s and t where cell s is initially set on fire and the fire expands in discrete time steps, will the target cell t eventually ignite? The new model builds on existing ones by introducing two integer parameters to each cell, which model the cell's resistance against ignition and its remaining fuel. This simple extension adds a lot of computational power to this model: We prove by reduction that the fire-expansion model is Turing-universal, i. e., it is capable of simulating the computations of every Turing machine. This implies that the initial question is undecidable on Turing machines, i. e., no general algorithm exists that can solve every instance of this problem within any finite number of steps. Since the fire-expansion model is related to concepts like cellular automata and artificial neural networks, both of these concepts are briefly reviewed and their relation to the model is discussed. The results of this chapter provide another example of a general observation in science and nature: Simple, local structures with a simple, local behaviour can often lead to incredibly complex global behaviour. Unlike in many other models, this is even possible if the cells cannot restore their initial state.

Organisation of this chapter. The remainder of this chapter briefly introduces the fundamental concepts and the formal notation used throughout this thesis. It indicates where these concepts are required in this thesis and provides references to related articles and textbooks for further reading. This chapter closes with a detailed overview of how the content of this thesis relies on results that were previously presented at conferences or workshops, and published in peer-reviewed proceedings or journals.

1.1 Graphs and Basic Exploration Algorithms

Many interesting search problems on graphs involve the search for a single or multiple targets. Often, it is assumed that a target can hide at any vertex of a graph $G = (V, E)$. Thus, a strategy might need to search at every vertex of G in the worst case. Moreover, if G is unknown from the beginning, the strategy can not compute a shortest walk to traverse G optimally; quite the contrary, the strategy has to systematically explore the vertices and edges of G. Hence, such a strategy is also

[3] For example, shortest-path or sorting problems.

called an *exploration* strategy for G. Two fundamental exploration algorithms for graphs follow the diametrically opposed search paradigms *Breadth-First Search* (BFS) and *Depth-First Search* (DFS): Both start at a specified vertex $s \in V$; while BFS gradually expands the search depth from s, DFS successively visits a non-marked neighbour of the current vertex if possible and retracts otherwise. For a more precise description of both graph algorithms see, e. g., [23]. Although both algorithms will eventually locate a stationary target in every finite graph, their performance can be quite different; often, the performance of these basic exploration algorithms can be improved in certain settings, see, e. g., [29, 56, 58, 64]. Finally, BFS and DFS are a starting point for many other applications; to give two examples: They can be used to test certain properties of G, e. g., whether G is connected or even a tree; moreover, BFS can be applied to compute a shortest path between vertices u, v of a directed acyclic graph, i. e., a u-v path of minimal weight.

Graphs, breadth-first search and depth-first search appear several times throughout this thesis. Section 3.2 introduces Kirkpatrick's [64] discrete search problem on lists, for which he developed a new measure of intrinsic complexity balancing BFS and DFS in an optimal way. In Chapter 4, we search for roundings of a finite, undirected, edge-weighted graph G. For the special case where G is a tree, such a rounding can be efficiently computed. Finally, the new fire-expansion model, introduced in Chapter 5, is based on a planar, node-weighted cell-graph. Moreover, the chapter considers a directed, infinite configuration-graph which models the behaviour of an abstract machine that can carry out computations of an arbitrary computer program. In such an infinite, directed configuration-graph neither BFS nor DFS can always find a desired target vertex t when starting from a given vertex s. As we will see, the situation is even worse: There exists no algorithm that can solve this problem in general!

The notation which we introduce in the following is commonly used in Mathematics and Computer Science; e. g., it can be found in [23, 65, 83]. A *graph* G is defined as a tuple (V,E) where V denotes a finite set of *vertices* and E a finite set of *edges*. Each element of E is an unordered pair of distinct vertices. Two vertices $u, v \in V$ are called *adjacent* if $\{u,v\} \in E$; thus, v is called a *neighbour* of u and the *neighbourhood* of u is the set of all neighbours of u. The *degree* of u is the number of neighbours of u. Graphs can also be defined for infinite sets of vertices and edges; we will explicitly distinguish between finite and infinite graphs where necessary.

In addition to these basic definitions, there are several other important definitions for structures in G. A graph is called *directed* if E is a binary relation on V, i. e., every element of E is an ordered pair of distinct vertices; to distinguish a directed from an undirected edge, we write (u,v) instead of $\{u,v\}$. In a directed graph, v is called a neighbour of u (or adjacent to u) if $(u,v) \in E$. The *outdegree* of a vertex u is the number neighbours of u and the *indegree* is the number of nodes to whom u is adjacent to. The triple (V,E,ω) is called a weighted graph G if ω is a mapping that assigns a (real) weight to every edge of E. A v_0-v_l walk π is a sequence of vertices $v_0,\ldots,v_l$ where $\{v_i,v_{i+1}\} \in E$ for every $0 \leq i < l$; v_0 is also called the start and v_l the end of π. The number of edges l is called the *length* of the walk and $\omega(\pi) \coloneqq \sum_{i=0}^{l-1} \omega(\{v_i,v_{i+1}\})$ its *weight*. A *path* is a v_0-v_l walk where $v_0,\ldots,v_l$ are distinct; a *cycle* is a walk where start and end are identical, i. e., $v_0 = v_l$. A graph is called *connected* if there exist a u-v walk for every pair of vertices $u, v \in V$.

There are many important types of graphs, also called graph classes; relevant in this thesis are the following ones. An undirected graph $T = (V,E)$ is called a *tree* if T is simply connected and does not contain a cycle; if all vertices of T have degree at most two, it is also called a *path graph* or *list*. A directed graph that does not contain a cycle is called a *Directed Acyclic Graph* (DAG). We call a graph *planar* if it can be drawn in the Euclidean plane where: vertices correspond to points in the plane, edges correspond to arcs between points, and arcs only touch at points that correspond to

vertices; for a more formal definition of planarity, see ,e. g., [83, 101]. Moreover, an undirected graph is called *regular* if all vertices have the same degree; a directed graph is called *regular* if all vertices have the same in- and outdegree. Finally, a regular, undirected graph is called a cell-graph if the nodes of the graph are cells which are joined by an edge if their cells lie side by side.

Certainly, a reason why graphs are a very popular discrete model is the simplicity with which many real world applications can be naturally modelled. In traffic networks, cities can be modelled as vertices whereas edges model connecting highways. In social networks, registered users can be modelled as vertices and an edge is created whenever two persons become friends or one follows the other. Last but not least, graphs can be used to model the behaviour of a physical machine, such as an elevator or a vending machine; the resulting graph is also called a state diagram.

1.2 Basic Concepts of Computational Geometry

Many interesting geometric search problems involve the search for points or paths in the Euclidean space. Problems where points in a certain subspace have to be located (and reported) usually ask for data-structures which support these operations efficiently; for further cf. [26, 65, 87]. Problems that ask for certain paths within a region R often consider *shortest paths* and *shortest escape paths*, where the term *shortest* refers to the length of the trajectory. While a shortest path between two points $s,t \in R$ does not need to intersect with ∂R, a *shortest escape path* for R must intersect with ∂R for every possible orientation and every possible start $s \in R$. In two-dimensions, shortest *s-t* paths can be computed efficiently in simple polygons [46, 76]. There are also efficient algorithms to compute shortest *s-t* paths in scenes with polygonal obstacles in the Euclidean plane [54]. However, in three-dimensional space, we cannot expect such an efficient solution; already the combinatorial problem of determining the sequence of obstacles-edges touched by a shortest path is NP-hard [18]. It seems even more difficult to find shortest escape paths for regions in the Euclidean plane; only for rather few types of regions a shortest escape path is known [36]. Moreover, there is no general algorithm to compute a shortest escape path for a given region. For a survey on geometric shortest path problems, see, e. g., [80]. In general, the search for paths (trajectories) with certain properties has been subject of many research articles and book in the field of competitive path planning; e. g., see [3, 59, 60].

Concepts of (Computational) Geometry appear several times throughout this thesis. Chapter 2 considers the search for a single target point t in a bounded subspace of the Euclidean space; e. g., the unit interval, the unit disk, the unit cube, or the unit circle. The search space considered in Chapter 3 is an unknown, simply connected region R in the Euclidean plane; the aim is to develop general strategies (geometric trajectories) which search for ∂R and approximate the unknown shortest escape path of R. We also present an algorithm to compute a certain escape path in simple polygons. Finally, in Chapter 5, we embed an infinite graph into the Euclidean plane; the geometric coordinates of the embedded vertices are used to store the register content of an abstract machine model.

The notation introduced in the following is commonly used in Mathematics and Computer Science; it can be found e. g. in [65]. For any fixed $n \in \mathbb{N}$, we call the metric vector space $(\mathbb{R}^n, d)$ equipped with the *Euclidean metric* $d : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$, $d(p,q) := \left(\sum_{i=1}^{n} (p_i - q_i)\right)^{1/2}$ the n-dimensional *Euclidean space*. For the special cases where $n = 1$ and $n = 2$, the search spaces are also called the *line* and the *Euclidean plane*. A subspace S of the Euclidean space is a subset $S \subset \mathbb{R}^n$ equipped with the Euclidean metric.

Often, the search space is restricted to a subspace of the Euclidean space. The *unit interval* is the subspace $[0,1]$ of the line; the n-dimensional *unit cube* denotes the subspace $[0,1]^n$ of $\mathbb{R}^n$. With

$B_\varepsilon(p)$, we denote the *open n-ball* of radius ε and center p, i. e., $B_\varepsilon(p) \coloneqq \{q \in \mathbb{R}^n \mid d(p,q) < \varepsilon\}$; moreover, with $B_\varepsilon[p] \coloneqq \{q \in \mathbb{R}^n \mid d(p,q) \leq \varepsilon\}$, we denote the *closed n-ball*. For $n = 2$, the subspace $B_1[0]$ is called the *closed unit disc* and the boundary $\partial B_1[0] \coloneqq \{q \in \mathbb{R}^2 \mid d(p,q) = 1\}$ of which is called the *unit circle*.

In general, a subspace $S \subset \mathbb{R}^n$ can be open or closed. A point p in a subspace S is called an *interior* point of S if an $\varepsilon > 0$ exists whereby $B_\varepsilon(p) \subset S$. Moreover, S is called *open* if all points of S are interior points; S is called *closed* if its complement, i. e., $S^c \coloneqq \{p \in \mathbb{R}^n \mid p \notin S\}$ is open. A point $p \in \mathbb{R}^n$ is called a boundary point of S if, for every $\varepsilon > 0$, $B_\varepsilon(p)$ contains a point of S and a point of S^c; the set of all boundary points of S is called the *boundary* of S and denoted with ∂S.

Important objects studied in Euclidean space are paths. A *path* γ is (the image of) a continuous and rectifiable mapping of $[0,1]$ into $\mathbb{R}^n$. We consider $\gamma(0)$ as the *start* of the path and $\gamma(1)$ as its *end*; γ is called *closed* if start and end coincide, i. e., $\gamma(0) = \gamma(1)$. With $\{\gamma\}$ we denote the *trace* of γ, i. e., $\{\gamma(t) : 0 \leq t \leq 1\}$, and with $|\gamma|$ we denote its *length*. Moreover, γ is called a *simple path* if all of its points, except for possibly start and end, are distinct; i. e., $\forall t,t' \in [0,1)$, $\gamma(t) = \gamma(t')$ implies $t = t'$.

Subspaces of the Euclidean plane with important properties are (simply connected) regions. A subspace $R \subset \mathbb{R}^2$ is called *path-connected* if, for every two points $s,t \in R$, there exists a path γ from s to t that fully lies in R, i. e., $\gamma(0) = s, \gamma(1) = t$ and $\{\gamma\} \subset R$. Moreover, R is called a *region* if R is open and path-connected. The *Jordan curve theorem* states that every simple, closed path γ subdivides the Euclidean plane into exactly two regions: An inner, bounded region with γ as its boundary; and an outer, unbounded region of which the boundary is also given by γ. Finally, R is called *simply connected* if R is path-connected and, for every simple, closed path γ in R, the inner, bounded region is fully contained in R.

Important objects studied in the Euclidean space or plane are line segments, polygonal chains and polygons. A *line segment* $\overline{pq}$ between two point $p,q \in \mathbb{R}^2$ is defined as the set of points $\{p + \lambda(q-p) \mid 0 \leq \lambda \leq 1\}$. For a sequence of points $p_1, \ldots, p_n$, the path γ which consist of the sequence of line segments $\overline{p_i p_{i+1}}$ for $1 \leq i < n$ is called a *polygonal chain*. If the polygonal chain γ is simple and closed, it defines a simply connected, closed region which is called a *(simple) polygon* P. Note that the boundary ∂P of P is exactly γ and belongs to P. The points $p_1, \ldots, p_n$ are also called the *vertices* (or corners) of P; the line segments $\overline{p_i p_{i+1}}$ are also called the *edges* of P. The *size* of P is given by the number n of vertices of P.

Other important objects are defined based on the concept of visibility. In an arbitrary region R, a point $q \in R$ is said to be *visible* from $p \in R$ if the line segment $\overline{pq}$ is fully contained in R, i. e., $\overline{pq} \subset R$. Moreover, the *kernel* of R is the set of points from which all other points in R are visible, i. e., $\ker(R) \coloneqq \{p \in R \mid \forall q \in R : \overline{pq} \subset R\}$. Finally, given a polygon P and a specific point $p \in P$, we define the *visibility polygon of p in P* as the subset $V_p \subset P$ of all points which are visible from p. Note that the visibility polygon V_p consists of some of the vertices and (parts of) the edges of P and can be computed efficiently, cf. [65].

Certainly, a reason why the Euclidean space is a popular model is the way it naturally models the space in which we live. Up to three dimensions, it is easy to visualize and gain intuition on a geometric problem. Moreover, the Euclidean space is equipped with a natural metric d that provides important properties: For any $p,q,r \in \mathbb{R}^n$, the distance from p to q is symmetric and non-negative, i. e., $d(p,q) = d(q,p) \geq 0$; it further satisfies the *triangle inequality* $d(p,r) \leq d(p,q) + d(q,r)$. While discrete models, such as graphs, often abstract from geometric information, this data can be maintained in Euclidean space. Often, it can be exploited to develop more efficient algorithms for a given problem. The field of Computational Geometry aims to use geometric information and

geometric properties to develop efficient algorithms for geometric problems; see [26, 65] for further reading.

1.3 Models of Computation and Computability

To examine computational problems theoretically, it is necessary to agree on a model of computations. These theoretical models make the results independent of existing technology, specific hardware and computer architectures; instead, they allow to focus on the mathematical nature of computational problems and to gain insight into their structure. Many different machine models have been developed in the past to capture the intuitive notion of *computability* or to address specific features of computational hardware.

Throughout this thesis, the following models of computation appear, more or less explicitly, several times. With respect to the runtime analysis of algorithms, we assume that the underlying machine model is a (variant of the) *Random-Access Machine* (RAM): In Chapter 2, this is a standard RAM; depending on whether the input is rational or real-valued, Section 4.3 presumes a RAM supporting the floor operation or the stronger real RAM model; definitely, a real RAM is presumed in Section 3.3.5. With respect to the polynomial-time reductions described in Section 4.2 and Section 5.3, the standard RAM model, which is equivalent to the Turing machine model, suffices to establish the connection between two computational problems. In addition to the RAM model, we use another machine model in Chapter 5: two-register machines. This simple model is capable of simulating the computations of every Turing machine, which we point out in Section 5.3.1. Moreover, we show how to simulate computations of two-register machines with a new fire-expansion model introduced in Chapter 5. The notation defined in the following is commonly used in Mathematics and Computer Science; it varies slightly between authors, e. g., see [4, 14, 100].

Turing machines are an important model for the study of computational problems. A deterministic *Turing machine* $M = (\Sigma, Q, \delta)$ is a tuple where Σ is a finite set of symbols, called *alphabet*; Q is a finite, non-empty set of possible states; and $\delta : Q \times \Sigma \to Q \times \Sigma \times \{-1, 0, +1\}$ defines a transition function. The tape consists of an infinite number of cells[4] and a *head* that always points to one of the cells. Each cell of the tape contains a single *character*, i. e., an element of Σ; all except for a finite number of cells contain the predefined *blank* symbol $\sqcup \in \Sigma$. Ignoring the infinite number of cells with blank symbols to the very left and right of the tape, we can describe the content of the tape with a finite sequence $\sigma = (\sigma_1, \ldots, \sigma_n)$ of characters which is called a *string* or *word* over Σ; moreover, n defines the *length* of σ. The *configuration of* M at a discrete time t is a snapshot of M's current state $q \in Q$, M's tape string σ, and the index i of the character to which the head currently points; thus, it can be uniquely represented with the tuple (σ, q, i). At any time t, a computational *step* is carried out by: (i) reading the character $a \in \Sigma$ to which the tape head points to and (ii) applying the transition function at time t to obtain $\delta_t(q, a) = (q', a', m)$. This transfers M into a state $q' \in Q$, replaces the character a at the head's position with a', and moves the tape head to the left ($m = -1$) or right ($m = +1$) neighbouring cell or not at all ($m = 0$). Thus, M is transferred from configuration (q, σ, i) to configuration $(q', \sigma', i + m)$.

Turing machines provide a formal definition of what we described intuitively as an algorithm at the beginning of this chapter. The *input* of a Turing machine M is a finite string $\sigma \in \Sigma^*$; the length $|\sigma|$ of the string is called the *input size*; we assume that the head initially points to σ_1. We say that M *terminates* (or halts) for the input σ if there exists a time $t > 0$ such that $\delta_t(q, a) = (q, a, 0)$ where

[4] More precisely, the tape has an infinite number of cells to the left and right direction. Thus, every cell has a unique left and right neighbouring cell.

δ_t denotes the t-th application of the transition function.[5] We define the *runtime* of M for σ as the minimum number of steps provided that the M terminates for input σ; otherwise we consider the runtime to be infinite. Since the runtime of M does not only depend on the input size but also on the specific symbols of σ, we often consider the *worst-case* runtime for an input of size n; i. e., the maximum runtime of M over all possible inputs of size n. The *output* of M is given by the final tape's string if M terminates.

Based on the definition of a Turing machine, we further define many other important terms. A Turing machine M uniquely identifies an input $\sigma \in \Sigma^*$ with an output $\sigma' \in \Sigma^*$ if M halts on σ, since M is deterministic; thus, M *computes* a partial function $f_M : \Sigma^* \to \Sigma^*$. To put it the other way round, we call a function f *computable* if there exists a Turing machine that computes f. We call two different Turing machines M, M' *equivalent* if they compute the same function; i. e., for every $\sigma \in \Sigma^*$, either M, M' both halt and $f_M(\sigma) = f_{M'}(\sigma)$ or neither of them halts. Although both compute the same function, M and M' may very well differ with regard to their runtime; hence, we might be interested to compare two Turing machines with regard to their runtime. For this purpose we introduce the following notation.

To compare the asymptotic growth of two functions, we use the following standard notation [44]. For two functions $f, g : \mathbb{N} \to \mathbb{R}_{\geq 0}$, we write $f \in \mathrm{O}(g) \Leftrightarrow \lim_{n\to\infty} {}^{f(n)}/_{g(n)} < \infty$ and $f \in \Omega(g) \Leftrightarrow g \in \mathrm{O}(f)$. Throughout this thesis, we use this notation not only to analyse the asymptotic runtime of algorithms; we use it also to bound the number of certain objects or the precision of an algorithm depending on the input size. Let $f(n)$ denotes the worst-case runtime of an algorithm with an input of size n; then we write that $f(n)$ is *polynomial* if there exists a polynomial $g(n)$ such that $f(n) \in \mathrm{O}(g(n))$. Note that the notation allows to simplify the polynomial g to a monomial with multiplicative coefficient one. Since these simplifications are of minor importance in the limit, we may speak of asymptotic bounds: We write *g is an upper bound for f* (g bounds f from above, g upper-bounds f) if and only if $f \in \mathrm{O}(g)$; similarly, we write *g is a lower bound for f* (or g bounds f from below, g lower-bounds f) if and only if $f \in \Omega(g)$.

One reason why the Turing machine is a popular model is certainly its simple and robust definition. Many modifications of the definition do not substantially change the model. For example, it makes no difference to use any fixed, positive number of tapes; use any fixed, positive number of heads per tape; use only tapes with a semi-infinite number of cells; reduce the size of the alphabet to only the digits 0 and 1. All these modified versions can be simulated by Turing machines of the above definition with a runtime overhead which is upper-bounded by a polynomial in the input size. Furthermore, Turing machines proved to be equivalent to several other attempts to define the intuitive notion of computability such as: general recursive functions; WHILE- and GOTO-programs; λ-calculus; RAMs; register machines. The fact that all these definitions are basically equivalent, supports the *Church-Turing thesis* which states that the functions computed by Turing machines are exactly those functions which can be computed intuitively; it further allows us to use the terms *computable* and *computability* independent of the Turing machine model.

Another reason for the popularity of the Turing machine model might be its practical implications. Certainly, the computations of a Turing machine can be easily carried out using pen and paper. However, the most stunning implication is that Turing machines are computers and algorithms at the same time; they also give a formal definition of what hard- and software is: Since all defining components of a Turing machine M are of finite size, M can be described as a finite string σ_M itself. Thus Turing showed that general purpose machines are conceivable which lead to the design of the

[5] Note that there will be no more changes to the tape's content in future steps. Alternative definitions assume that TM switches to a special halting state.

first, modern computers. Moreover, Turing showed that a so-called *Universal Turing machine U* exists which is able to carry out the operations of any other Turing machine M given in the form of σ_M. This Universal Turing machine U can be seen as a sort of (low-level) hardware computer which is capable of running any given software program. Furthermore, U is a key component to prove the *undecidability* of problems such as the *halting problem*, i. e., no algorithm exists which can decide whether a Turing machine will ever halt.

Another important model for the study of computational problems is the *Random-Access Machine* (RAM). Compared to Turing machines, this model is slightly more oriented to the design of real computers. The RAM consists of an unlimited number of registers each of which can contain an arbitrarily large integer or floating point number. To manipulate the content of a register, the RAM executes *instructions*, one at a time. The set of available instructions usually contains arithmetic operations such as additions, subtractions, multiplications or divisions; data-movement operations to load, store and copy register content; and operations to manipulate the control flow such as conditional statements, jumps, call of subroutines or return statements.

The RAM model simplifies the analysis of algorithms tremendously. Often, algorithms are given in *pseudocode*, a (textual) high-level description of the operations that an algorithm needs to perform. Given the pseudocode of an algorithm, its runtime can be analysed by simply counting the number of basic operations a RAM has to carry out. Compared to the basic operations provided by Turing machines, this simplifies the runtime analysis for the algorithm essentially. Luckily, an analysis based on the RAM model is not fundamentally different as when based on the Turing machine model: One can prove that a RAM can be simulated by a Turing machine with only a small polynomial runtime overhead and vice versa.[6]

Frequently used variants of the standard RAM are the *real RAM* and the *word RAM* model. The real RAM extends the standard definition by the following two assumptions: First, every register can store an arbitrarily large *real* number; second, the set of operations contains analytic functions such as k-th root, exp, log and trigonometric functions. Since its first description in [91], this model has become the standard model in the field of Computational Geometry [26, 65, 87]. In the word RAM model, the size of every register is limited to a certain number of bits. For example, if the input consists of integer values with a maximum absolute value of $n \in \mathbb{N}_0$, the number of bits per register is assumed to be $c \cdot \log(n)$ for some constant $c \geq 1$. This avoids that huge amount of data can be stored in a single register.

Functions can be related to each other using so-called reductions. The notation introduced in the following is inspired by Blum [14] and lectures of Rolf Klein; it is slightly more general than other definitions and allows to relate functions which depend on different models of computation. Let $f : \Sigma^* \to \Sigma^*$ and $g : \Gamma^* \to \Gamma^*$ be two (partial) functions. We define that *f can be reduced to g* if and only if there exists a pair of functions $r, r' : \Sigma^* \to \Gamma^*$ which relates the input and the output of f to g such that the diagram on the right commutes; moreover, we assume that r is total and r' is injective. Thus, for every input $\sigma \in \Sigma^*$ it is $r'(f(\sigma)) = g(r(\sigma))$ and $f(\sigma)$ can be obtained by applying the (partial) inverse function of r' to $g(r(\sigma))$, alternatively. Note what such a reduction implies for f: If the functions r, g, r' are Turing-computable, so is f and we call the reduction a *Turing-reduction*; moreover, if r, g, r' are computable in polynomial time on Turing machines each,

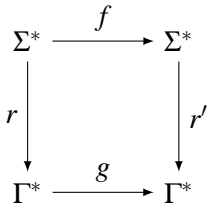


Figure 1.2: Reducing f to g.

[6] This holds in the logarithmic cost model where the cost of a RAM operation depends on the number of bits used to store the corresponding arguments. For a proof, cf. [14] or [4].

so is f and we call the reduction a *polynomial-time reduction*. Since f and g solve computational problems, we also write that a problem is *polynomial-time reducible* to another problem without explicitly mentioning the underlying functions. Finally, note that f and g could rely on completely different models of computation; e. g., in Chapter 5 we prove that every Turing-computable function f can be reduced to a function g which is defined by a fire's expansion in the new fire-expansion model.

1.4 Basic Concepts of Computational Complexity

The model of computation is useful to classify (decision) problems with respect to their resource requirements, e. g., their worst-case runtime. Such classifications are subject to Complexity theory. A fundamental problem of Computational Complexity arises from propositional logic and asks for the satisfiability of Boolean formula.

Boolean expressions, the complexity classes P and NP, and the satisfiability problem SAT appear several times throughout this thesis. In Section 2.2, we use Boolean expressions to optimize a given query strategy: In a preliminary step, we model minimum requirements as a Boolean formula; then, we convert the formula to DNF to read off optimal solutions. The complexity classes NP and the satisfiability problem SAT appear explicitly in Chapter 4 of this thesis; by polynomial-time reduction, SAT is reduced to the decision problem as to whether an ε-rounding of a given edge-weighted graph and an error-threshold ε exists. The reduction implies NP-hardness for the rounding problem.

In the following, we summarize only basic concepts of Complexity theory using standard notation and the Turing machine model; for further reading and a more detailed description we refer to [4]. A subset $L \subset \Sigma^*$ is called a *language* over the alphabet Σ; the total mapping $\chi_L : \Sigma^* \to \{0,1\}$ with $\chi_L(\sigma) = 1 \Leftrightarrow \sigma \in L$ is called the *characteristic function* of L or a *decision problem* in general[7]. We write that a Turing machine M_L *decides* L if M_L computes the characteristic function χ_L.

The complexity class P is the set of languages which can be decided by Turing machines in polynomial time; i. e., $L \in \mathrm{P}$ if χ_L is computable in $\mathrm{O}(|\sigma|^c)$ time for every input $\sigma \in \Sigma^*$ and some constant $c \in \mathbb{N}$. Similarly, the class NP is the set of languages for which their characteristic function can be *verified* by Turing machines in polynomial time; i. e., for every language $L \in \mathrm{NP}$ there exists a function $f_L : \Sigma^* \to \{0,1\}$ with the following property: For every $w \in L$, there exists a $v \in \Sigma^*$ such that $f_L(v) = 1$ and f_L is computable in time $\mathrm{O}(|w|^c)$ by Turing machines for some constant $c \in \mathbb{N}$.[8] Note that the definition of P and NP is robust with respect to the machine model; more precisely, it does not depend on whether a Turing machine or RAM is the underlying model of computation.

Based on the hypothesis that P is a proper subset of NP, reductions are the key to interrelate different problems and thus define so-called *complexity classes*, i. e., sets of problems that can be decided with similar computational effort. This has been done in the 1960s by defining the complexity classes P and NP with the aim to explain why some computational problems can be solved efficiently on modern computers whereas others cannot. It has been continued with the discovery of the polynomial time (and space) hierarchy and numerous other complexity classes. Recently, reductions have been used to refine problems within P in order to explain the lack of improvements in theoretical runtime bounds for various popular problems over many years.

Important problems in Computational Complexity arise from propositional logic. A *Boolean expression* or *formula* α consist of variables $y_1, \ldots, y_n$, the logical operators $\wedge, \vee$ and $\neg$, and brackets $(,)$. To each variable, one can assign either the logical value `true` or `false` which we represent with

[7] This stems from the fact that χ_L maps a word in L to a truth value and thus *decides* whether a given $\sigma \in \Sigma^*$ is part of L.

[8] Note that the polynomial runtime bound on the runtime of f_L implies a polynomial bound on the length of v.

the binary values 1 and 0. An *assignment* for α is a mapping $\psi : \{y_1, \ldots, y_n\} \to \{0,1\}$. A Boolean formula α is said to be in *conjunctive normal form* (CNF) if α can be written in the form

$$\alpha = \bigwedge_{1 \le i \le m} c_i \quad \text{with} \quad c_i := \left(\bigvee_j l_{ij} \right),$$

where each *literal* l_{ij} is either a variable y_k or its negation $\overline{y}_k$. Each disjunction c_i is called a *clause* of α. Similarly, α is said to be in *disjunctive normal form* (DNF) if α can be written in the form

$$\alpha = \bigvee_{1 \le i \le m'} c'_i \quad \text{with} \quad c'_i := \left(\bigwedge_j l_{ij} \right).$$

Note that a clause of α in DNF is a conjunction of literals.

A key problem in NP is the Boolean satisfiability problem of propositional logic (SAT). The problem SAT asks whether a given CNF formula α is satisfiable; more precisely, SAT is the language of Boolean expressions which are in CNF and have a satisfying assignment. Cook and Levin independently proved that any other problem in NP can be reduced to an instance of SAT in polynomial time. Thus, SAT cannot be solved substantially faster than any other problem in NP; it is said to belong to the set of hardest problems in NP which is also called NP-hard problems. Certainly, SAT is in NP since the correctness of a given assignment can be verified by simply evaluating the formula.

Bibliographical notes

This thesis is based on research carried out during my time as a PhD student at Department I and V at the University of Bonn between 2015 and 2020. For each chapter, the following overview specifies which of the content was previously published at peer-reviewed conferences and journals; it indicates which colleagues and co-authors were involved and how the content differs from these publications.

Chapter 2 Research on this topic was initiated by Herman Haverkort at the workshop *Search Games: Theory and Algorithms* at the *Lorentz Center* in Leiden in 2016. Several participants were involved in finding initial solutions; among them, Endre Csóka and Bengt Nilsson. Further research has been carried out with Herman Haverkort, Elmar Langetepe and Barbara Schwarzwald. A first abstract was presented at the *European Workshop on Computational Geometry (EuroCG) 2017* [51]. Major results were published and presented at the *Algorithms and Data Structures Symposium (WADS) 2017* [52]. A full version of this paper was published in the journal *Computational Geometry: Theory and Applications* [53].

This thesis contains unpublished material: Optimal query strategies for small numbers of query points; a comprehensive presentation of how query strategies were developed starting from these initial results; algorithms in pseudocode for query strategies on the line; a closed form expression for the recursive formula used in the incremental strategy; a detailed discussion of the online incremental setting and a variants where the search space is the unit circle.

Chapter 3 Research on the approximation of shortest escape paths and certificate paths was initiated by Elmar Langetepe. First major results are due to Elmar Langetepe and were published and presented

at the *Scandinavian Symposium and Workshops on Algorithm Theory (SWAT) 2016* [74]. An *arXiv preprint* of the conference version is also available [75].

The conference version has been extended in several ways: A comparison of the certificate path with Isbell's shortest escape path for the infinite strip; new results concerning the approximation of shortest escape paths for families of regions with a non-empty kernel; enhanced proofs for the approximation of certain shortest escape paths via the certificate path; a detailed proof for the approximation lower bound. A full version of this extended paper was submitted to the journal *Computational Geometry: Theory and Applications*.

Additionally, this thesis contains unpublished material: An algorithm to compute the certificate path for arbitrary starting positions within simple polygons; an analysis of how the certificate path can be approximated with a strategy of expanding circles; detailed checks that the functionals satisfy the preconditions of the Theorem of Gal; an improved approximation lower bound which closes the gap between upper and lower bound; a discussion of how the lower bound technique can be applied to the problem of 'searching for a point in the plane with a radar' or other geometric search problems in general.

Chapter 4 Research on this topic was initiated by Herman Haverkort and has been carried out together with Herman Haverkort and Elmar Langetepe. Major results were published and presented at the *International Workshop on Combinatorial Algorithms (IWOCA) 2019* [49]. A full version of the conference paper is available as an *arXiv preprint* [50].

In addition to the conference version, this thesis contains a discussion of how the existence of an ε-rounding depends on the choice of ε. Moreover, the range of choices of ε which imply NP-hardness, has been slightly extended as compared to the conference version.

Chapter 5 Research on the new model started during a research visit by Jörg-Rüdiger Sack at our department. It has been carried out together with Rolf Klein, Elmar Langetepe, Jörg-Rüdiger Sack and Barbara Schwarzwald. Initial results were published and presented at the *Annual International Conference on Algorithms and Discrete Applied Mathematics (CALDAM) 2020* [67]. A full version of the conference paper is available as an *arXiv preprint* [66]. Moreover, some of these results are presented in the doctoral thesis of Barbara Schwarzwald [90].

This thesis contains the following unpublished material. In contrast to previous publications, this thesis focuses on the computability aspects of the new fire-expansion model. It contains a full proof that every Turing machine can be simulated with constant time overhead in a two-layer version of the fire-expansion model; a sketch of this proof is already contained in [66]. The thesis discusses also implications of this result. Moreover, since cellular automata and artificial neural networks are related to the fire-expansion model, this thesis reviews both concepts and discusses their relation to the new model.

Chapter 2

Searching for a Stationary Target Point

This chapter examines the following search problem in a *continuous* search space: A *single* target point t hides in a bounded part of the Euclidean space. To pinpoint the location of t as precise as possible, we issue query points $q_1, \ldots, q_n$ in the search space; the feedback is an ordering of the query points by distance to t.

With respect to the feedback, we distinguish two variants: Either query points have to be placed all at once, which we refer to as the *one-shot variant*; or, points are placed one by one depending on the response received on the previous queries which we refer to as the *incremental variant*. For both variants we establish lower and upper bounds on the precision that can be achieved. At first, we consider the case where the search space is the unit interval or the unit circle, i. e., the boundary of the unit disc. Then, we discuss what this implies for higher dimensional search spaces.

2.1 Formal Problem Statement and Related Work

In many settings one essentially searches for a target based on information retrieved from an ordering or ranking of objects. Imagine we want to set up receivers to locate an animal which carries a tracking device that sends signals. The strength of the signals may vary, but we know one thing for sure: the closer the animal is, the stronger is the signal. Or imagine a researcher conducting a survey who wants to summarize respondents' political preferences by scoring them on several scales, e. g., from conservative to progressive, or from favouring a small state to a large state. Respondents may not be able to score themselves but, given a number of hypothetical party programmes, they can rank them and say which one they like best.

To address settings like these, we consider the following model. We assume that the target is a point in a one- or higher-dimensional space. To pinpoint the location of the target, we issue queries (receivers, party programmes) that are points in the same configuration space. As a response, we obtain an ordering of the query points according to their distance to the target. From this we try to derive the location of the target with the highest possible accuracy; or conversely, we try to reach high accuracy with as few (expensive) queries as possible.

Searching for a stationary target is a common problem in Computer Science and Applied Mathematics. Strategies such as evenly distributed query points or binary search immediately spring to mind; however, as we will see in this chapter, at least in certain abstract settings of the problem we can do much better. The problem of efficiently obtaining an order on a set of objects has been studied before in very general settings [62], but note that in our case, the cost measure is the number of query points that are used, not the number of comparisons that are made between them. The reconstruction

of geometric objects based on a sequence of geometric probes, such as points, lines, hyperplanes, or wedges, has also been investigated: the problem was introduced by Cole and Yap [21] and the main focus is also on the number of queries [94].

Specifically, we focus on the following setting.

Definition 1 (query strategy). *Given a bounded subspace R of the Euclidean space, a certain number of queries $n \in \mathbb{N}$ and a target t at an unknown position within R.*

A query point *is a specific point in R and a* query strategy *is an ordered set of n query points $q_1, \ldots, q_n \in R$. The* response *of the i-th query is an ordering of the points $q_1, \ldots, q_i$ by increasing Euclidean distance to t.*

The response of a query allows to restrict possible locations of the target t within R to a subspace of R. Given two query points q_i, q_j, the *bisector* of q_i and q_j is the set of points for which the distance to q_i equals the distance to q_j, which we denote with $\mathrm{bs}(q_i, q_j)$. For every pair of query points, their bisector divides R into two subspaces. Thus, the ordering of the query points restricts possible locations of t to one of these two subspaces. Considering all pairs of query points, possible locations of t are restricted to a subspace bounded on each side by a bisector or the boundary of R.

In the following, we use the diameter of the resulting subspace as the measure for the accuracy. One can argue to use the area of the region instead; however, this can result in very thin and long regions contradicting the aim to precisely locate the target, as the diameter limits the area but not vice versa.

Definition 2 (accuracy). *Let* $\mathrm{diam}(R)$ *denote the diameter of R and let $S \subset R$ be the subspace in which the target is found to lie after the n-th query point has been placed.*

The accuracy *of a query strategy is the minimum value* $\mathrm{diam}(R)/\mathrm{diam}(S)$ *over all possible locations of $t \in R$.*

With respect to the frequency of the responses, we distinguish two variants. In the *one-shot variant*, the response is only given after all n query points $q_1, \ldots, q_n$ have been placed. In this case one needs to maximize the number of different, well-spaced bisectors. Such combinatorial questions are classical problems in discrete geometry; see for example [77]. Moreover, this variant is closely related to problems from Arithmetic Combinatorics or Combinatorial Number Theory. In the *incremental variant*, a response is given after each query point placement and may affect the choice of the next point. Thus, query strategies can operate in a binary search fashion, but we can do even better than that, as we will see later. Note that the incremental variant can also be interpreted as a game where an adversary tries to hide the target in the largest possible area. Geometric games about area optimization have a tradition in Computational Geometry, e. g., see [2].

Organisation of this chapter. In the remainder of this section, we present research related to our search problem. Section 2.2 examines the one-shot variant of the search problem on the line. We prove a quadratic lower bound for the accuracy that any strategy can achieve and provide a non-trivial strategy with accuracy $\Theta(n^2)$, which is at most a factor two from optimal. Section 2.3 examines the incremental variant of the search problem on the line. We develop a strategy with accuracy $\Omega(2.29^n)$ and prove that no strategy can achieve accuracy $\Omega(3.66^n)$. In Section 2.2, we discuss the one-shot and incremental variant for other search spaces, such as the unit circle, the unit disc, and n-dimensional unit cube and sphere. We conclude with Section 2.5.

Related Work

As already mentioned above, query strategies in the one-shot variant are related to problems from Arithmetic Combinatorics and Combinatorial Number Theory. This field of mathematics seeks to develop estimates associated with arithmetic operations such as addition, subtraction, multiplication, and division.

Erdős and Turán [30, 31] and, in 2010, Cilleruelo et. al. [20] developed bounds for the *Sidon sequence* problem [93]. This problem, named after Fourier analyst Simon Sidon, considers positive, increasing, integer sequences which are bounded by some value $N \in \mathbb{N}$. The challenge is to find such a sequence $\mathscr{A}$ that contains a maximum number of elements under the restriction that all pairwise sums of elements of $\mathscr{A}$ have to be different. Please note that *not* every integer $x \in [1,N]$ has to have a representation as the sum of two elements of $\mathscr{A}$; however, if x is the sum of two elements, this representation has to be unique.

The same problem has been independently studied by different researchers in various contexts and under different names. Bloom and Golomb [13] used the notion of *rulers*: A so-called *Golomb ruler* is a set of marks along an integer ruler, such that no two marks have the same distance. Formally, a Golomb ruler is a set $\mathscr{A} = \{a_1, a_2, \ldots, a_n\}$, such that $a_i - a_j = a_k - a_l \Leftrightarrow i = k \wedge j = l$ for $1 \leq i, j, k, l \leq n$. Thus, it is not difficult to see that every Golomb ruler is a Sidon set and vice versa. More variants of these rulers have been considered; probably most relevant to mention with regard to our query problem is the *m-perfect ruler of length n*. For this type of Golomb ruler, every x up to a certain $m \leq N$ equals the difference of exactly one pair of elements.

How Sidon sequences and Golomb rulers are related to query strategies for the one-shot variant of our problem are discussed at the end of Section 2.2.

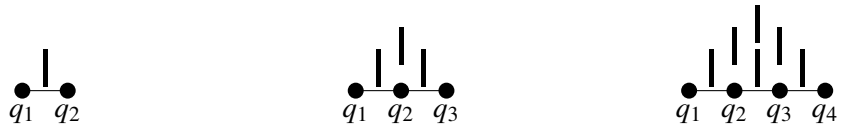
2.2 One-Shot Strategies on the Line

In this section, we consider the one-shot variant of the problem and restrict the search space R to an interval on the line. W. l. o. g. we assume that R is the unit interval $[0,1]$; by scaling and shifting, every query strategy can be adapted to work for an arbitrary search interval $[l,r]$.

In the one-shot variant we get the response only after placing all n points. This ordering pinpoints the target t to a subinterval bounded by bisectors of query points or by the boundary of the unit interval. As the target may lie in any subinterval, the problem is equivalent to minimizing the maximum size of such a subinterval. As n query points can produce at most $\binom{n}{2} = 1/2 \cdot (n^2 - n)$ distinct bisectors, there are at most $1/2 \cdot (n^2 - n) + 1$ subintervals. Thus, we get the following (trivial) upper bound for the accuracy of one-shot strategies:

Theorem 1. *No deterministic strategy using n query points in the unit interval can locate an arbitrary target with accuracy larger than* $\frac{1}{2} \cdot (n^2 - n + 2) \in O(n^2)$ *in the one-shot variant.*

Surprisingly, there is a query strategy with an accuracy that is very close to this upper bound. Instead of only presenting the strategy and proving its accuracy, we develop this strategy throughout this section: In Section 2.2.1, we discuss optimal query strategies for cases where the number of query points is small and derive a simple general strategy from these observations; then, this query strategy is improved in Section 2.2.2 to achieve an accuracy of at least $1/4 \cdot (n^2 + 6n - 27)$.



(i) $n = 2$ (ii) $n = 3$ (iii) $n = 4$

Figure 2.1: Optimal one-shot query strategies for the case where $n \leq 4$. The query points of EQUIDIST are given as labelled dots on the unit interval in the bottom line. Every black rectangle above the line indicates a bisector $\mathrm{bs}(q_i, q_j)$ generated by q_i, q_j; the closer q_i and q_j are along the line, the closer the rectangle is to the line.

2.2.1 Developing a Simple Query Strategy from Optimal Strategies for $n \leq 4$

If the number of query points is small, optimal query strategies can be easily found. Obviously, for $n = 1$, no bisector subdivides the unit interval; consequently, the accuracy can not be increased at all and is equal to one. Using two query points q_1, q_2, exactly one bisector $\mathrm{bs}(q_1, q_2)$ is created; consequently, the search space can be split into at most two subintervals of equal size limiting the accuracy for $n = 2$ to two. A similar argument can be applied to bound the maximum accuracy for $n = 3$: Three query points generate at most three distinct bisectors dividing the unit interval into four subintervals to achieve an accuracy of at most four. For $n = 2$ and $n = 3$, the optimal strategies illustrated in Figure 2.1i and ii match these lower bounds.

Looking only at the optimal query strategies for $n \leq 3$, one might hope that the accuracy can be doubled with every additional query point; however, due to Theorem 1, this is impossible. In fact, this is already impossible for $n = 4$ as the following observation shows. The idea of this proof was already mentioned by Endre Csóka at the workshop *Search Games: Theory and Algorithms* at the Lorentz Center in Leiden in 2016.

Observation 1. *The accuracy of an optimal one-shot strategy using $n = 4$ query points is* 6.

Proof. W. l. o. g., we may assume that $0 \leq q_1 \leq q_2 \leq q_3 \leq q_4 \leq 1$. Let the accuracy of the query strategy be $1/a$, then the maximum size of a subinterval is a; this bounds the positions of the two left most bisectors $\mathrm{bs}(q_1, q_2), \mathrm{bs}(q_1, q_3)$ and the right most bisector $\mathrm{bs}(q_3, q_4)$ from which we derive the following inequalities.

$$\mathrm{bs}(q_1, q_2) \leq a \quad \Rightarrow \quad q_2 \leq 2a - q_1 \quad \leq \quad 2a \tag{2.1a}$$

$$\mathrm{bs}(q_1, q_3) \leq \mathrm{bs}(q_1, q_2) + a \quad \Rightarrow \quad q_3 \leq q_2 + 2a \quad \overset{(2.1a)}{\leq} \quad 4a \tag{2.1b}$$

$$1 \leq \mathrm{bs}(q_3, q_4) + a \quad \Rightarrow \quad 1 \leq q_3 + q_4 + 2a - 1 \quad \overset{(2.1b)}{\leq} \quad 6a \tag{2.1c}$$

Equation 2.1c implies that the accuracy $1/a$ is at most 6; in fact, this accuracy can be achieved choosing $(q_1, q_2, q_3, q_4) = (0, 1/3, 2/3, 1)$, see Figure 2.1iii. □

One can also prove this observation via linear programming. For this purpose, one can assign a variable to each query point and force them to lie in ascending order by introducing appropriate constraints. The objective is to minimize the size of the largest subinterval. To express the size of the subintervals in terms of the variables, one needs to fix the ordering of the bisectors using the constraints of the linear program. Due to symmetry, one can restrict the attention to only one such

ordering for $n = 4$; however, for larger values of n, this is no longer true. Thus, to prove an upper bound for $n > 4$, one rather requires a technique different to linear programming.

Observation 1 shows two things: First of all, the accuracy cannot be doubled with every additional query point. Second, it seems like a good idea to place one query point at each end of the interval and to place the remaining ones equally spaced; thus, one achieves optimal accuracy for the cases where $n \in \{2,3,4\}$. Let us denote this strategy applied to the unit interval with EQUIDIST(n) in the following; moreover, let $\mathscr{Q}_n \coloneqq \{q_1,\dots,q_n\}$ denote the set of query points generated by the strategy. Algorithm 1 gives pseudocode for the general case where the search interval is $[l,r]$.

Algorithm 1: EQUIDIST$([l,r],n)$

Input: An interval $[l,r]$ with $l < r$ that contains the target point t; a number of query points $n \geq 2$.
Output: A set $\mathscr{Q}_n$ of n equally spaced query points $l = q_1 < q_2 < \cdots < q_n = r$.

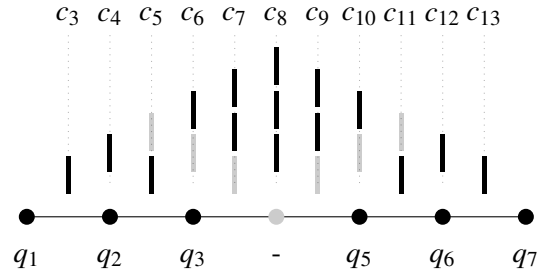
```
dist ← (r − l)/(n − 1)
for i ← 0 to (n − 1) do
    add q_{i+1} := (l + i · dist) to 𝒬
return 𝒬
```

It is not difficult to prove that the accuracy of EQUIDIST(n) is exactly $2(n-1)$. In a first step, one can prove the following statement via induction over n: $\mathscr{Q}_n$ generates bisectors only at positions $q_2,\dots,q_{n-1}$ and $1/2 \cdot (q_i + q_{i+1})$ for $1 \leq i < n$; i. e., at every query point q_i and between every consecutive pair of query points q_i, q_{i+1}. Finally, the accuracy of EQUIDIST(n) follows from the fact that there are exactly $2n-2$ subintervals of equal size; the subintervals are generated by $2n-3$ equally spaced bisectors from which $n-2$ bisectors lie exactly on query points $q_2,\dots,q_{n-1}$, and $n-1$ many lie between every consecutive pair.

2.2.2 Improving the Accuracy of EQUIDIST

As we have seen before, EQUIDIST achieves optimal accuracy for $n \in \{2,3,4\}$; however, this is no longer true for $n \geq 7$ as the following observation shows: Removing q_4 from $\mathscr{Q}_7$ does not change the accuracy. For a graphical proof, we refer to Figure 2.2; the figure also illustrates that the number of redundant, that is to say coinciding, bisectors generated by EQUIDIST(n) grows with the number of query points n. Thus, the fact that some query points of $\mathscr{Q}_n$ can be removed without loss of accuracy is due to redundancy among the bisectors. Unfortunately, for larger values of n it is not easy to see which of the query points could be removed; luckily, we can use the following approach to find a solution with the help of a computer.

For every set of query points $\mathscr{Q}_n$ generated by EQUIDIST(n), we can construct a Boolean formula α_n with the following property: α_n can be satisfied by setting at most k variables to `true` if and only if $n-k$ points of $\mathscr{Q}_n$ can be removed without loss of accuracy. As mentioned above, $\mathscr{Q}_n$ generates bisectors only at positions $q_2,\dots,q_{n-1}$ and $1/2 \cdot (q_i + q_{i+1})$ for $1 \leq i < n$; for ease of description, we call these positions *columns* which we denote with $c_3,\dots,c_{2n-1}$, see Figure 2.2i. Thus, a bisector in column c_k is generated by every pair of distinct query points q_i, q_j where $i + j = k$; moreover, a subset $\mathscr{S} \subset \mathscr{Q}_n$ achieves the same accuracy as $\mathscr{Q}_n$ if and only if the points in $\mathscr{S}$ generate a bisector in every column. We assign a Boolean variable y_i to every query point $q_i \in \mathscr{Q}_n$ and let a subset $\mathscr{S}$ correspond to an assignment of the variables in the following way: y_i is `true` if $q_i \in \mathscr{S}$ and `false` otherwise. For each column c_k, we create the disjunction $c_k \coloneqq \bigvee_{i+j=k} (y_i \wedge y_j)$ where each conjunction $(y_i \wedge y_j)$ models the bisector generated by q_i and q_j if y_i, y_j are both `true`; the disjunction



(i) Removing q_4 from the set of points $\mathscr{Q}_7$.

$$\begin{aligned}\alpha_7 \;\coloneqq\;& (y_1 \wedge y_2) \wedge (y_1 \wedge y_3)\\ \wedge\;& \big((y_1 \wedge y_4) \vee (y_2 \wedge y_3)\big)\\ \wedge\;& \big((y_1 \wedge y_5) \vee (y_2 \wedge y_4)\big)\\ \wedge\;& \big((y_1 \wedge y_6) \vee (y_2 \wedge y_5) \vee (y_3 \wedge y_4)\big)\\ \wedge\;& \big((y_1 \wedge y_7) \vee (y_2 \wedge y_6) \vee (y_3 \wedge y_5)\big)\\ \wedge\;& c_9 \wedge c_{10} \wedge c_{11} \wedge c_{12} \wedge c_{13}\end{aligned}$$

(ii) Corresponding Boolean expression for $\mathscr{Q}_7$.

Figure 2.2: Careful removal of query points generated from EQUIDIST(n) can give a strategy for fewer points but with the same accuracy: (i) q_4 has been removed from $\mathscr{Q}_7$, the set of query points generated by EQUIDIST(7). As before, bisectors are indicated with rectangles above the line: Grey rectangles indicate bisectors which q_4 would generate with every other query point; in every column at least one bisector (black rectangle) remains. (ii) α_7 can be satisfied by setting only six variables (all except y_4) to `true`.

of conjunctions models the constraint that at least one bisector in the column is required to satisfy c_k. Finally, we obtain α_n as the conjunction $\alpha_n \coloneqq \bigwedge_{3\le k\le 2n-1} c_k$ since a bisector is required in every column to maintain the accuracy; an example for $n = 7$ is given in Figure 2.2ii. Altogether, a subset $\mathscr{S} \subset \mathscr{Q}_n$ corresponds to an assignment of the variables and satisfies α_n if and only if $\mathscr{S}$ achieves the same accuracy as $\mathscr{Q}_n$.

To remove as much redundancy among the bisectors of $\mathscr{Q}_n$ as possible while maintaining the accuracy, one can equivalently seek for satisfying assignments of α_n that sets a minimum number of variables to `true`. For this purpose α_n can be converted into disjunctive normal form (DNF) in a first step; in a second step, one can determine the minimum length l_n of a conjunction in DNF and remove all conjunctions which are strictly longer than l_n. Both steps can be carried out with the help of computer programs such as Mathematica or Maple; resulting formulas are given in Table 2.1 for several small values of n. Every minimal subset $\mathscr{S} \subset \mathscr{Q}_n$ which achieves the same accuracy as $\mathscr{Q}_n$ is of size l_n; moreover, if α_n is in DNF, the elements of such a subset $\mathscr{S}$ can be read off the variables in any conjunction of length l_n.

The assignments given in Table 2.1 reveal a pattern for removing query points to minimize redundancy. It is always necessary to set the first (last) three variables to `true`, which implies that the three query points to the left (right) are never removed. For $l_n < 11$, the fourth query point from left (and from right) is removed; several of the remaining points are removed to leave regular gaps of size at most two. However, for $l_n = 11$, one observes that leaving gaps of size at most two is no longer a unique optimal solution. In fact, keeping four query points from left and right, it is possible to leave regular gaps of size three[1]; moreover, for $l_n \ge 12$ leaving gaps of size two is no longer optimal! This suggests that the size of the gaps should be increased with the number of query points. In general, keeping the first $(x+1)$ query points from the left (and right) allows to leave regular gaps of size x.

The removal of query points can be used to define a strategy for a given number of n query points. This strategy chooses a subset of size n among the query points of EQUIDIST$(\varphi_x(n))$ where $\varphi_x(n)$ is

[1] Both possibilities to remove a maximum number of query points from $\mathscr{Q}_{21}$ while maintaining the accuracy are also illustrated in Figure A.1 in the appendix.

n	l_n	α_n in DNF trimmed to clauses of length l_n
7	6	$(y_1 \wedge y_2 \wedge y_3 \wedge y_5 \wedge y_6 \wedge y_7)$
8	6	$(y_1 \wedge y_2 \wedge y_3 \wedge y_5 \wedge y_6 \wedge y_7 \wedge y_8) \vee (y_1 \wedge y_2 \wedge y_3 \wedge y_4 \wedge y_6 \wedge y_7 \wedge y_8)$
15	9	$(y_1 \wedge y_2 \wedge y_3 \wedge y_5 \wedge y_8 \wedge y_{11} \wedge y_{13} \wedge y_{14} \wedge y_{15})$
18	10	$(y_1 \wedge y_2 \wedge y_3 \wedge y_5 \wedge y_8 \wedge y_{11} \wedge y_{14} \wedge y_{16} \wedge y_{17} \wedge y_{18})$
21	11	$(y_1 \wedge y_2 \wedge y_3 \wedge y_5 \wedge y_8 \wedge y_{11} \wedge y_{14} \wedge y_{17} \wedge y_{19} \wedge y_{20} \wedge y_{21}) \vee$ $(y_1 \wedge y_2 \wedge y_3 \wedge y_4 \wedge y_7 \wedge y_{11} \wedge y_{15} \wedge y_{18} \wedge y_{19} \wedge y_{20} \wedge y_{21})$

Table 2.1: For a given number of query points n in the first column, the second column gives the minimum length l_n for clauses of α_n in DNF. The third column gives α_n in DNF trimmed to clauses of length l_n; each of these clauses corresponds to a minimal subset $\mathscr{S} \subset \mathscr{Q}_n$ which achieves the same accuracy as $\mathscr{Q}_n$.

an appropriate function of n and the maximum gap size x. Specifically, the strategy $\textsc{Gap}_x(n)$ uses only the following n points from $\textsc{EquiDist}(\varphi_x(n))$:

- The first $(x+1)$ query points to the left, i. e., $q_1, q_2, \ldots, q_{x+1}$;
- the last $(x+1)$ query points to the right, i. e., $q_{\varphi_x(n)-x}, \ldots, q_{\varphi_x(n)-1}, q_{\varphi_x(n)}$;
- every $(x+1)$-th query point starting with q_{2x+1}, i. e., $q_{2x+1+k\cdot(x+1)}$ for $k \in \{0, \ldots, n-(2x+3)\}$.

This results in widely spaced query points throughout the search range, and tightly spaced points near both ends, omitting at most x consecutive query points from $\textsc{EquiDist}(\varphi_x(n))$. Most useful bisectors will then be formed by one of the tightly spaced points and one of the widely spaced points.

It remains to show how to compute φ. Let us assume that $x > 0$ and n is sufficiently large (more precisely, $n > 2x+4$). Then, all kept and removed points can be added up; using that n query points remain in the end, we obtain

$$\varphi_x(n) = \underbrace{2 \cdot \big(2x+1\big)}_{(i)} + \underbrace{x}_{(ii)} + \underbrace{\big(n-2(x+2)\big) \cdot \big(x+1\big)}_{(iii)} = n(x+1) - 2x^2 - x - 2,$$

where: (i) counts all (kept and removed) query points left of the first, and right of the last gap of size x; (ii) counts the removed query points of the first gap of size x to the left; and (iii) counts all occurrences of a gap of size x with a single query point to the right.

Lemma 1. *For $x, n \in \mathbb{N}$ with $n \geq 2x+3$, the one-shot query strategy $\textsc{Gap}_x$ has accuracy $2(\varphi_x(n)-1)$.*

Proof. It suffices to show that every distinct bisector of $\textsc{EquiDist}(\varphi_x(n))$ is also created by $\textsc{Gap}_x(n)$, as the points chosen by $\textsc{Gap}_x(n)$ form a subset of those chosen by $\textsc{EquiDist}(\varphi_x(n))$. Hence, for each $1 \leq i < \varphi_x(n)$, $\textsc{Gap}_x(n)$ must choose two points that form a bisector in the middle between q_i and q_{i+1}. For each $1 < j < \varphi_x(n)$, $\textsc{Gap}_x(n)$ must choose two points that form a bisector directly on q_j.

For $i \leq x$ and $j \leq x$ this is given by $q_1, q_2, \ldots, q_{x+1}$. For $x < i < \varphi_x(n)/2$, $\textsc{EquiDist}(\varphi_x(n))$ forms a bisector between q_i and q_{i+1} with all of the pairs $(q_1, q_{2i}), (q_2, q_{2i-1}), \ldots, (q_{x+1}, q_{2i-x})$. One of these pairs has to be in the chosen subset of $\textsc{Gap}_x(n)$, since the first $q_1, \ldots, q_{x+1}$ are all present, and the

choosing process omits at most x consecutive points from $q_{2i-x},\dots,q_{2i}$. Likewise, one of the pairs (q_1,q_{2j-1}), $(q_2,q_{2j-2}),\dots,(q_{x+1},q_{2j-(x+1)})$ must have been chosen by $\textsc{Gap}_x(n)$. Those pairs form a bisector on q_j for $x < j \leq \varphi_x(n)/2$. The existence of the remaining bisectors, that is, those in the right half of the unit interval, follows by the symmetry of the strategy.

Hence $\textsc{Gap}_x(n)$ and $\textsc{EquiDist}(\varphi_x(n))$ produce the same set of $2\varphi_x(n)-3$ distinct equidistant bisectors which gives an accuracy of $2(\varphi_x(n)-1)$ for both strategies. □

Given n, it remains to choose x optimally in order to maximize $\varphi_x(n)$. As the slope of the linear function $\varphi_x(n)$ increases with x, $\textsc{Gap}_{x+1}$ eventually supersedes $\textsc{Gap}_x$. The break-even point between $\textsc{Gap}_x$ and $\textsc{Gap}_{x+1}$ is at $n = 4x+3$; hence, the optimal choice is $x = \lceil (n-3)/4 \rceil$.

Note that the n query points of $\textsc{Gap}_x(n)$ can also be computed directly without the need to generate all $\varphi(n)$ points beforehand and remove all except for n afterwards; hence, $\textsc{Gap}_x(n)$ can be implemented to run in linear time, pseudocode is given in Algorithm 2.

Theorem 2. *For any number of points $n \geq 3$ and $x = \lceil (n-3)/4 \rceil$, the one-shot strategy $\textsc{Gap}_x(n)$ has an accuracy of $2(\varphi_x(n)-1) \geq 1/4 \cdot (n^2+6n-27) \in \Omega(n^2)$.*

Considering the lower and upper bound given by Theorem 2 and Theorem 1, one observes that both bounds lie a small constant factor of two apart. As shown in Observation 1, it is (already) impossible to place four query points in such a way that their bisectors are all distinct and divide the unit interval into parts of equal size; hence, we conjecture that the trivial upper bound of Theorem 1 is not tight. Yet, it might also be possible to improve the lower bound; with our optimization approach we obtain a strategy which is only optimal among strategies with equidistant bisectors.

Algorithm 2: $\textsc{Gap}([l,r],n)$

Input: An interval $[l,r]$ with $l < r$ that contains the target point t; a number of query points $n \geq 2$.
Output: A set $\mathcal{Q}$ of n query points $l = q_1 < q_2 < \cdots < q_n = r$.

if $n \leq 5$ **then**
  $\textsc{EquiDist}([l,r],n)$
$x \leftarrow \lceil (n-3)/4 \rceil$
$\varphi_x(n) \leftarrow n \cdot (x+1) - 2x^2 - x - 2$
$\text{dist} \leftarrow (r-l)/(\varphi_x(n)-1)$
for i from 0 to x **do**
  add $q_{i+1} := (l + i \cdot \text{dist})$ and $q_{n-i} := (r - i \cdot \text{dist})$ to $\mathcal{Q}$
for k from 0 to $n-(2x+3)$ **do**
  add $q_{(x+2)+k} := ((l + 2x \cdot \text{dist}) + k \cdot (x+1) \cdot \text{dist})$ to $\mathcal{Q}$

On the Relation between One-Shot Strategies and Sidon Sequences/Golomb Rulers

As pointed out in Section 2.1, Sidon sequences and Golomb rulers are equivalent forms of the same problem. Recall that a Sidon sequence $\mathcal{A}$ is a positive, increasing integer sequence up to a largest value $N \in \mathbb{N}$ with the following constraint: For every integer $x \in [1,N]$, there is *at most* one representation of x as the sum of two elements of $\mathcal{A}$.

Query strategies in the one-shot variant seem closely related to Sidon sequences. This impression comes from the fact that redundancy in the pairwise sums is completely eliminated for Sidon sequences; however, this is not true for the $\textsc{Gap}$ strategy, where redundancy in the bisectors remains.

The strategy GAP selects a subset $\mathscr{S} \subset \mathscr{Q}_n$ such that: For every $k \in [3, 2n-1]$, there is *at least* one bisector in column c_k, which is formed by elements of $\mathscr{S}$. Another difference to Sidon sequences is that query points are not restricted to lie on integer (or rational) coordinates. Placing query points on real-valued coordinates or even in an asymmetric way, one might be able to achieve higher accuracy.

Regardless of these differences, it may very well be that results from Arithmetic Combinatorics or Combinatorial Number Theory can help to narrow the gap between the lower and upper bound.

2.3 Incremental Strategies on the Line

In this section we consider the incremental variant of the problem and restrict the search space R to an interval on the line; w. l. o. g. we assume that R is the unit interval. In the incremental variant, we learn the ordering of $q_1, \ldots, q_{i-1}$ by increasing distance to the target point t before placing query point q_i; thus, we may choose the location of q_i depending on this information. As in the one-shot variant, the ordering of the query points pinpoints the location of t to a subinterval bounded by bisectors of query points or the boundary of R. The overall aim is to minimize the size of the subinterval in which t is found to lie after placing the last query point q_n.

It is not difficult to see that a strategy querying R in a binary search manner achieves an accuracy of 2^{n-1}. This strategy could place the first two query points at 0 and 1 to increase the accuracy to two; the remaining query points can be placed such that the interval containing t is halved in each step and the accuracy is doubled. Surprisingly, we can do better than this with a recursive strategy that takes advantage of more than one new bisector in many steps as we will see later.

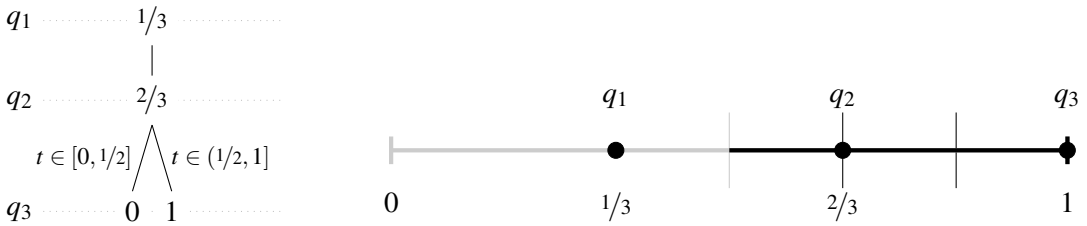
A first attempt to bound the accuracy of every incremental query strategy gives a (trivial) upper bound of $(n!)$. Let R_{i-1} denote the subinterval of R in which the target is known to lie after placing q_{i-1}. Placing query point q_i, we obtain $(i-1)$ new bisectors that q_i generates with previously placed query points. These bisectors split R_{i-1} into at most i subintervals where at least one of which must have size at least $1/i$ times the size of R_{i-1}. Thus, the accuracy increases with a factor of at most i leading to an accuracy of at most $(n!)$ after q_n has been placed.

Another attempt to bound the accuracy of incremental strategies is to relate incremental to one-shot strategies and apply Theorem 1, the (trivial) upper bound of one-shot strategies. In fact, an incremental query strategy for n query points defines a query tree of height $(n-1)$: Each node of this tree specifies where q_i has to be placed depending on the interval in which t is found to lie, see e. g. Figure 2.3i or Figure 2.4i. Using all query points of the query tree at once, one obtains a one-shot strategy that achieves the same accuracy. It remains to bound the size of the query tree: Similar to the above reasoning, placing q_i creates $(i-1)$ new bisectors which split the interval into at most i new subintervals; hence, a node of the tree in height i has at most $(i+1)$ children. This adds up to at most $\sum_{i=0}^{n-1} i!$ nodes in total and proves the following observation.

Observation 2. *For every incremental strategy with n queries, there exists a one-shot strategy with at least the same accuracy using at most* $\sum_{i=0}^{n-1} i!$ *query points.*

Unfortunately, none of these two approaches to bound the accuracy of optimal incremental strategies from above is tight. Note that the second approach does not even give a better bound than the first, since the upper bound of Theorem 1 introduces another quadratic factor. Nevertheless, the first bound is useful to prove optimality of certain incremental strategies for small values of n.

The remainder of this section is organized as follows. We begin with discussing optimal incremental query strategies for small numbers of query points in Section 2.3.1; the ideas of these query strategies are used to improve on the lower and upper bound in subsequent sections. In Section 2.3.2,



(i) Query tree. (ii) Example query for $t \in (0.5, 1]$.

Figure 2.3: The query tree in (i) defines an optimal incremental query strategy for $n = 3$: After placing q_1, q_2, their bisector splits the interval in two subintervals of equal size; q_3 can be placed either at 0 or 1, depending on the subinterval in which the target t is found to lie. The choice for q_3 further splits the subinterval into three parts of equal size, as shown in (ii). This results in an accuracy of 6.

we prove the existence of a query strategy that achieves an accuracy of $\Omega(2.29^n)$ and thus outperforms the binary search approach. The proof already shows how this strategy can be obtained. For completeness, we give query algorithms in Section 2.3.3. In Section 2.3.4, we show that the naive $(n!)$ upper bound for the accuracy of optimal incremental strategies can be improved to single exponential growth; more precisely, we prove an upper bound of $O(3.66^n)$.

2.3.1 Optimal Query Strategies for $n \leq 4$

Obviously, for $n = 2$, optimal incremental strategies are identical to optimal one-shot strategies and achieve accuracy of at most 2.

For $n = 3$, an optimal query strategy with accuracy 6 is illustrated in Figure 2.3. The bisector of q_1, q_2 splits the interval in half which increases the accuracy by a factor of two. Then, q_3 is used to generate two more bisectors which further split a half into three subintervals of equals size; this gives an accuracy gain of three. Optimality of this strategy follows immediately from the $(n!)$ bound or Observation 2.

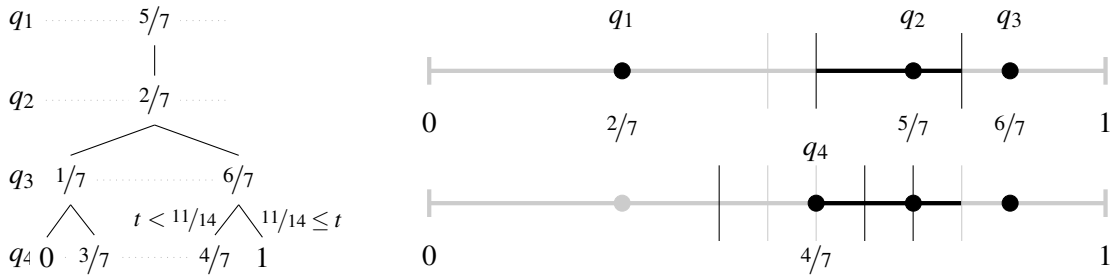
For $n = 4$, a query strategy with accuracy 14 is illustrated in Figure 2.4. Similar to the strategy for three query points, the bisector of q_1, q_2 splits the interval in half; however, the bisectors generated by q_3 now split each half into one small and two larger subintervals; by placing q_4, each of the larger subintervals can be split into three parts of equal size matching the size of the small subinterval. In the third step, the gain of accuracy is slightly larger than two; it is exactly three for the larger subintervals in the fourth step.

Especially the query strategy for $n = 3$ gives hope that the accuracy can be increased by a factor of three from step three onward. From this point of view, it feels like the strategy for $n = 4$ is suboptimal and has some room for improvement: First of all, the two bisectors generated by q_3 do not split the interval into three subintervals of equal size; moreover, one of the bisectors generated by q_4 is not even used to increase the accuracy. This feeling is supported by the fact that $4! = 24$ only gives a rather loose upper bound for the accuracy.

As the following observation shows, such large accuracy gains are already impossible for $n = 4$. Since the proof of this observation is rather long and technical, it is given in Section A.2.

Observation 3. *The accuracy of an optimal incremental strategy using $n = 4$ query points is 14.*

We end this section with a closer look at the query strategy given in Figure 2.4 in order to gain some insight into characteristics of strategies for $n > 4$. There are two important features to observe:



(i) Query tree.

(ii) Example query for $t \in (4/7, 11/14]$.

Figure 2.4: The query tree in (i) defines an incremental query strategy for $n = 4$. Compared to the strategy of Figure 2.3, the choice for q_3 splits each half into three subintervals of *different* size: One subinterval of size $1/14$, which is already small enough; two subintervals of size $3/14$, which are further divided into thee parts by placing q_4, as illustrated in (ii). This results in an accuracy of 14.

First of all, it might be possible to use more than one bisector in many steps to improve the accuracy even if an interval is not split into parts of equal size; this idea ultimately led to the discovery of the incremental query strategy presented in Section 2.3.2. Second, one of the bisectors generated by q_4 lies outside the current subinterval (no matter where q_4 is placed); hence, some of the newly created bisectors in a step might not be able to contribute to increasing the accuracy. More precisely, two previously placed query points need to lie close enough so that they can (possibly) contribute to accuracy gains when the next query point is placed; this observation lead to the concept of close pairs which we use to prove a general upper bound for the accuracy of incremental query strategies in Section 2.3.4.

2.3.2 A Query Strategy with High Accuracy

In the following, we develop a strategy that takes advantage of at least two bisectors in many steps, except for the first. As we will see, this leads to a mean increase of accuracy of more than a factor two per step. Consequently, this strategy outperforms the binary search approach for $n \ge 3$. Let $h(n)$ be the accuracy we aim for, as a function of n, the number of query points we get to place. For ease of description, we use the interval $[0, h(n)]$ instead of the unit interval and pinpoint the target to an interval of length at most 1.

Similar to the optimal strategies for $n \le 4$, our incremental strategy places the first two query points symmetrically to split the initial interval $[0, h(n)]$ in half.[2] Let $g(n)$ denote the distance of the first two query points to their common bisector, so q_1 is placed at $\frac{1}{2}h(n) - g(n)$ and q_2 at $\frac{1}{2}h(n) + g(n)$. Now suppose the target lies to the right of the bisector, the other case is symmetric. We now place q_3 at some distance $2f(n)$ to the right of q_2 and find that the target lies in one of three intervals, see Figure 2.5a:

(I) $[\mathrm{bs}(q_1, q_2), \mathrm{bs}(q_1, q_3)] = [\frac{1}{2}h(n), \frac{1}{2}h(n) + f(n)]$, of size $f(n)$;

(II) $[\mathrm{bs}(q_1, q_3), \mathrm{bs}(q_2, q_3)] = [\frac{1}{2}h(n) + f(n), \frac{1}{2}h(n) + g(n) + f(n)]$, of size $g(n)$; or

(III) $[\mathrm{bs}(q_2, q_3), h(n)] = [\frac{1}{2}h(n) + g(n) + f(n), h(n)]$, of size $\frac{1}{2}h(n) - g(n) - f(n)$.

[2] In fact, an optimal strategy places q_1, q_2 always like this as discussed in the proof of Observation 3.

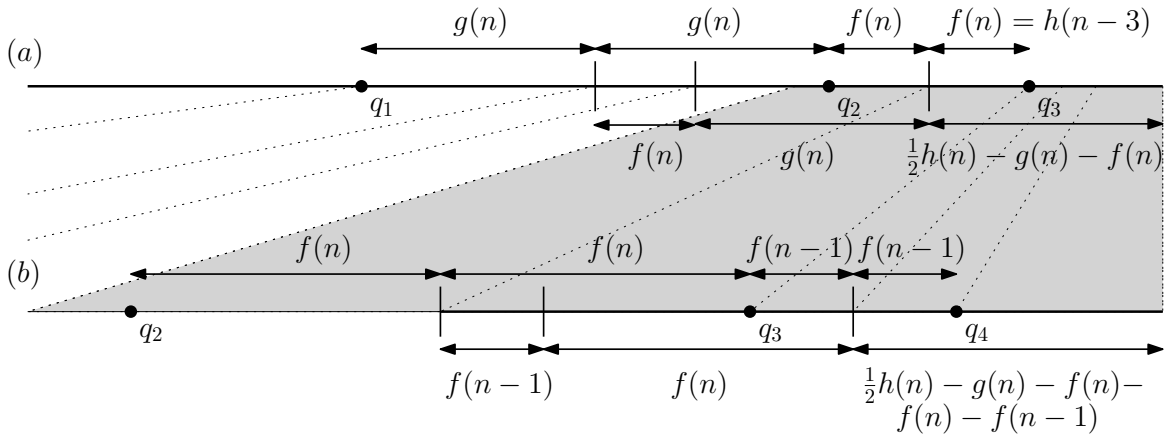


Figure 2.5: Placement of the first query points for the incremental query strategy locating a target with accuracy $h(n)$.

To interval (I) we apply the strategy recursively with the remaining $n-3$ query points, starting from scratch; hence, we have to choose $f(n) = h(n-3)$. In contrast, for the intervals (II) and (III) we exploit the previously placed query points q_2 and q_3 at distance $f(n)$ left and right of their bisector. We describe the strategy in the following paragraph only for interval (III); the same strategy can be applied symmetrically to the second interval provided that the second interval is not larger than the third for $n \geq 3$, which we refer to as condition (A). Note that condition (A) implies $g(n) \leq 1/2 \cdot h(n)$, which ensures that q_1 and q_2 lie within the interval $[0, h(n)]$. To ensure $q_3 \in [0, h(n)]$, we also require $g(n) + 2f(n) \leq 1/2 \cdot h(n)$ for $n \geq 3$, which we refer to as condition (B).

We now describe how our strategy queries interval (III) with two predetermined query points q_2, q_3 and $n-3$ query points $q_4, \ldots, q_n$ that can be chosen freely, see Figure 2.5b. We place q_4 right of q_3 at distance $2f(n-1) = 2h(n-4)$; to ensure that $q_4 \in [0, h(n)]$, this requires that $h(n-3) + 2h(n-4)$ is not larger than the size of (III) for $n \geq 4$ which we refer to as condition (C). Again, we find that the target lies in one of three subintervals, see Figure 2.5b:

(i) $[\mathrm{bs}(q_2, q_3), \mathrm{bs}(q_2, q_4)]$, of size $f(n-1)$;

(ii) $[\mathrm{bs}(q_2, q_4), \mathrm{bs}(q_3, q_4)]$, of size $f(n)$; or

(iii) $[\mathrm{bs}(q_3, q_4), h(n)]$, of size $\frac{1}{2}h(n) - g(n) - 2f(n) - f(n-1)$.

To subinterval (i) we apply the overall strategy recursively from scratch. Subinterval (iii) has predetermined query points q_3, q_4 at distance $f(n-1)$ left and right of its left boundary; hence, we can apply the adapted strategy recursively using the remaining $n-4$ points provided that its size follows the recursive definition, i. e., $1/2 \cdot h(n) - g(n) - 2f(n) - f(n-1) = 1/2 \cdot h(n-1) - g(n-1) - f(n-1)$ for $n \geq 4$ which we refer to as condition (D). The subinterval (ii) has predetermined query points at distance $f(n-1)$ left and right of its right boundary; we can apply the strategy recursively as long as it is smaller than the third subinterval, which gives $f(n) \leq 1/2 \cdot h(n-1) - g(n-1) - f(n-1)$ for $n \geq 4$ and is referred to as condition (E).[3]

Since $f(n) = h(n-3)$, it remains to choose h and g as functions of n such that

[3] To apply the query strategy of the third (sub-)interval to the second, we can enlarge the size of the second (sub-)interval to match the size of the third; explicitly, we do this in the algorithms in Section 2.3.3. To be always able to enlarge this interval, we have to verify two conditions: First, interval (I) and (II) are together larger than (III). Second, subinterval (i)

- the conditions for recursive application are satisfied, i. e., for $n \geq 4$ we have

  (A) $g(n-1) \leq \frac{1}{4}h(n-1) - \frac{1}{2}h(n-4)$;
  (B) $g(n-1) \leq \frac{1}{2}h(n-1) - 2h(n-4)$;
  (C) $g(n) \leq \frac{1}{2}h(n) - 2h(n-3) - 2h(n-4)$;
  (D) $\frac{1}{2}h(n) - g(n) - 2h(n-3) = \frac{1}{2}h(n-1) - g(n-1)$;
  (E) $g(n-1) \leq \frac{1}{2}h(n-1) - h(n-3) - h(n-4)$;

- and, when no further recursion is possible because we run out of query points, the interval containing the target is small enough: $h(0) \leq 1$, $h(1) \leq 1$ and $h(2) \leq 2$ for the initial interval and interval (I); and $1/2 \cdot h(3) - g(3) - f(3) \leq 1$ for interval (III), which also bounds the size of interval (II) by condition (C).

Theorem 3. *There is a deterministic, incremental strategy to locate a target using n query points in a unit interval with accuracy $\Omega(b^n)$, where $b > 2.29$ is the largest real root of $b^4 - b^3 - 6b - 2$. The accuracy $h(n)$ is given by the recurrence formula $h(0) = 1$, $h(1) = 1$, $h(2) = 2$, $h(3) = 6$ and $h(n) = h(n-1) + 6h(n-3) + 2h(n-4)$ for $n \geq 4$.*

Proof. We prove that, for the above conditions (A) to (E) and the restrictions on the size of the final interval, we can find suitable recursive starting values and definitions of g, h. Thus, by construction, we obtain a strategy of accuracy $h(n)$.

To start with, we show how to obtain recursive definitions of g and h that satisfy the conditions (A) to (D). Since h is a non-decreasing function of n, condition (B) is redundant, as it is already implied by condition (E). Rewriting condition (D) to $g(n) = g(n-1) + 1/2 \cdot (h(n) - h(n-1)) - 2h(n-3)$ and substituting it into condition (C) reveals that condition (C) is equivalent to condition (B), hence redundant. We now satisfy condition (A) by simply choosing equality $g(n-1) = \frac{1}{4}h(n-1) - \frac{1}{2}h(n-4)$ for $n \geq 4$. Substituting this in the remaining conditions we get the following: Condition (D) becomes $h(n) = h(n-1) + 6h(n-3) + 2h(n-4)$ for $n \geq 4$, which is the recurrence formula in the theorem; condition (E) becomes $h(n-1) \geq 4h(n-3) + 2h(n-4)$; using $g(n-1)$ this is equivalent to $h(n-3) \leq g(n-1)$. As the proof that this form of condition (E) holds for our choice of $g(n)$ is rather technical, we take care of it in Lemma 2 at the end of this section.

Finally, we prove that the final intervals are small enough when running out of query points. Obviously, this holds by definition of $h(n)$ for $n \in \{1,2,3\}$ and it remains to show that $\frac{1}{2}h(3) - g(3) - h(0) \leq 1$: We derive suitable values for $g(3)$ and $h(3)$ using $g(3) = \frac{1}{4}h(3) - \frac{1}{2}h(0)$ which simplifies this inequality to $h(3) \leq 6$; hence, we may choose $g(3) = 1$ and $h(3) = 6$. Note that the alternative form of condition (E) is already satisfied for $n = 4$, since $h(1) \leq g(3)$,

A closer analysis of the recurrence formula, the subsection on a closed form expression below, reveals that $h(n) \in \Omega(b^n)$, where $b \approx 2.29\ldots$, the largest root of the polynomial $b^4 - b^3 - 6b - 2$. □

Note that, for sake of simplicity, the above strategy is not quite optimal. If we proceed recursively with only few points left to place, we can sometimes do slightly better by taking some already placed points into account. For example, $h(2) = 2$ because, if we get to place only two points, we can only create one bisector and therefore we cannot achieve accuracy better than 2. However, if we start with more than 3 points and get to a final interval in which we get to place the last two points, then

and (ii) are together larger than (iii). For the first condition, this can be easily verified using the definition of $g(n)$ as given in Theorem 3. The second condition can be verified using $g(n) \geq 2g(n-1)$, which can be proven by induction over n itself.

each one can form a bisector together with a previously placed query point outside the interval. This allows us to assume $h(2) = 3$ in the recursive definition of $h(n)$ for $n \geq 4$, which slightly improves the constant factor in the asymptotic bound.

Finally, it remains to prove that the alternative form of condition (E) holds.

Lemma 2 (condition (E)). *If we choose $h(n)$ and $g(n)$ as in the proof of Theorem 3, then, for any $n \geq 4$, we have $h(n-3) \leq g(n-1)$.*

Proof. To prove the inequalities via induction over $n \geq 5$, we require some additional inequalities:

$$\begin{aligned} g(n) &\geq h(n-2) && (2.2)\\ 2h(n-3) &\leq h(n-2) && (2.3)\\ 2h(n-2)+h(n-3) &\geq h(n-1) && (2.4)\\ g(n) &\leq h(n-1)/2 && (2.5) \end{aligned}$$

Obviously, (2.2), (2.3), (2.4) and (2.5) hold for $n = 5$.[4] We make use of the following two recurrence formulas

$$\begin{aligned} h(n) &= 4g(n)+2h(n-3) && (2.6)\\ h(n) &= h(n-1)+6h(n-3)+2h(n-4) && (2.7) \end{aligned}$$

where (2.6) holds for $n \geq 3$ and (2.7) holds for $n \geq 4$ by definition. Substituting $h(n-1) = 4g(n-1)+2h(n-4)$ for $n \geq 4$ in (2.7) gives $h(n) = 4g(n-1)+6h(n-3)+4h(n-4)$ and substituting this formula for the left-hand side of (2.6) gives

$$g(n) = g(n-1)+h(n-3)+h(n-4) \tag{2.8}$$

for $n \geq 4$. Additionally, from (2.6) we conclude $h(n-4) = h(n-1)/2 - 2g(n-1)$ for $n \geq 4$. Substituting this into (2.8) for $h(n-4)$ gives

$$g(n) - h(n-1)/2 = h(n-3) - g(n-1) \tag{2.9}$$

for $n \geq 4$.

We perform the inductive step from n to $n+1$ for $n \geq 5$. We would like to show that under the assumption that (2.2), (2.3), (2.4) and (2.5) hold for an arbitrary, but fixed $n \geq 5$, we conclude that (2.2), (2.3), (2.4) and (2.5) hold for $n+1$. We additionally apply (2.6), (2.8) and (2.9).

$$\begin{aligned} (2.2)' \quad g(n+1) &= g(n)+h(n-2)+h(n-3) && \text{(From (2.8))}\\ &\geq 2h(n-2)+h(n-3) && \text{(From (2.2))}\\ &\geq h(n-1) && \text{(From (2.4))} \end{aligned}$$

$$\begin{aligned} (2.3)' \quad h(n-1)/2 &\geq g(n) && \text{(From (2.5))}\\ h(n-1) &\geq 2g(n) && (\Leftrightarrow)\\ &\geq 2h(n-2) && \text{(From (2.2))} \end{aligned}$$

$$\begin{aligned} (2.4)' \quad h(n) &= 4g(n)+2h(n-3) && \text{(From (2.6))}\\ &\leq 2h(n-1)+2h(n-3) && \text{(From (2.5))}\\ &\leq 2h(n-1)+h(n-2) && \text{(From (2.3))} \end{aligned}$$

[4] Note that (2.4) does not hold for $n = 4$ but we start the induction for $n = 5$.

$$(2.5)' \quad h(n)/2 \quad \geq \quad g(n+1) \quad \text{(From (2.9) for } n+1 \text{ and (2.2))}$$

Thus, condition (E) holds by induction. □

A Closed Form Expression for the Recurrence Formula $h(n)$

In the following, we derive a closed form expression for the recurrence formula $h(n)$ using the technique of generating functions, e. g., see [44]. In a first step, we use the recurrence relation[5] of Theorem 3 to define the semi-infinite integer sequence $(h_n)_{i\geq 0}$ where $h_0 := h_1 := 1, h_2 := 3, h_3 := 6$ and

$$h_n := h_{n-1} + 6h_{n-3} + 2h_{n-4} \quad \text{for } n \geq 4. \tag{2.10}$$

Then, we identify the elements of the sequence with coefficients of a power series to obtain the generating function $H(z) := \sum_{n\geq 0} h_n \cdot z^n$. Applying the recursive definition (2.10) to the coefficients of the series and manipulating the power series, we can derive the closed form

$$H(z) = \frac{3z^3 - 2z^2 - 1}{2z^4 + 6z^3 + z - 1} =: \frac{P(z)}{Q(z)}.$$

The final step involves to expand the closed form of $H(z)$ into a power series and read off the coefficients of h_n. For this purpose, we use that $Q(z)$ can be factorized using its four roots

$$r_1 = \left(-\tfrac{3}{4} - \tfrac{1}{2}\sqrt{\beta} - \tfrac{1}{2}\sqrt{\tfrac{27}{4} - \beta + \tfrac{31}{4\sqrt{\beta}}}\right) \quad r_2 = \left(-\tfrac{3}{4} - \tfrac{1}{2}\sqrt{\beta} + \tfrac{1}{2}\sqrt{\tfrac{27}{4} - \beta + \tfrac{31}{4\sqrt{\beta}}}\right)$$

$$r_3 = \left(-\tfrac{3}{4} + \tfrac{1}{2}\sqrt{\beta} - \tfrac{1}{2}\sqrt{\tfrac{27}{4} - \beta - \tfrac{31}{4\sqrt{\beta}}}\right) \quad \overline{r_3} = \left(-\tfrac{3}{4} + \tfrac{1}{2}\sqrt{\beta} + \tfrac{1}{2}\sqrt{\tfrac{27}{4} - \beta - \tfrac{31}{4\sqrt{\beta}}}\right)$$

into $Q(z) = 2(r_1 - z)(r_2 - z)(r_3 - z)(r_4 - z)$ where $\beta := \frac{9}{4} + \frac{\alpha}{2\cdot 3^{2/3}} - \frac{7}{\sqrt[3]{3}\cdot\alpha}$ and $\alpha := \sqrt[3]{\sqrt{31641} - 153}$. We apply the rational expansion theorem [44, p. 340] to obtain the complex-valued closed form expression

$$h_n = \frac{a_1}{r_1^n} + \frac{a_2}{r_2^n} + \frac{a_3}{r_3^n} + \frac{a_4}{\overline{r_3}^n} \quad \text{where} \quad a_i := \frac{-P(r_i)}{r_i \cdot Q'(r_i)}$$

with $Q'(z) := \frac{d}{dz}Q(z) = 8z^3 + 18z^2 + 1$. The coefficients can be evaluated to obtain approximately $a_1 \approx 0.57089, a_2 \approx 0.51391$ and $a_3 \approx -0.04240 + 0.01187i$. It is not difficult to prove that $a_4 = \overline{a_3}$ using the equivalences $\overline{z_1 + z_2} = \overline{z_1} + \overline{z_2}, \overline{z_1 \cdot z_2} = \overline{z_1} \cdot \overline{z_2}$ for conjugate complex numbers $z_1, z_2 \in \mathbb{C}$. Applying the same rules, we can finally derive a real-valued closed form for h_n as follows

$$\begin{aligned} h_n &= \frac{a_1}{r_1^n} + \frac{a_2}{r_2^n} + \frac{a_3}{r_3^n} + \frac{\overline{a_3}}{\overline{r_3}^n} = \frac{a_1}{r_1^n} + \frac{a_2}{r_2^n} + \left(\frac{a_3}{r_3^n}\right) + \overline{\left(\frac{a_3}{r_3^n}\right)} = \frac{a_1}{r_1^n} + \frac{a_2}{r_2^n} + 2 \cdot \Re\left(\frac{a_3}{r_3^n}\right) \\ &= a_1 \cdot \left(\frac{1}{r_1^n}\right) + a_2 \cdot \left(\frac{1}{r_2^n}\right) + 2 \cdot |a_3| \cdot \cos(\varphi_{a_3} - n \cdot \varphi_{r_3}) \cdot \left(\frac{1}{|r_3|^n}\right) \\ &\approx 0.57 \cdot (-0.32)^n + 0.51 \cdot 2.29^n + 2 \cdot 0.04 \cdot \cos(2.86 + 1.87n) \cdot 1.63^n, \end{aligned}$$

[5] And the fact that we may choose $h_2 = 3$ if $n \geq 4$, as mentioned above.

where $\varphi_{a_3} = \arg(a_3) \approx 2.86873$ and $\varphi_{r_3} = \arg(r_3) \approx -1.87329$ denote the arguments of the complex numbers a_3 and r_3 respectively.

2.3.3 A Recursive, Incremental Query Algorithm

In the following, we give pseudocode for the recursive query strategy described in Theorem 3. In each step, the position of the next query point depends on the number k of query points that remain to be placed and the current subinterval in which the target t hides. Thus, it is not necessary to compute the whole query tree for all possible locations of t in advance and the runtime of the algorithm is linear in the overall number of query points n. The description of the algorithm is based on the routine QUERYINTERVAL with two subroutines QUERYLEFTSUBINTERVAL and QUERYRIGHTSUBINTERVAL. The input of each routine is the current search interval $[l,r]$ in which t is known to lie and the number k. Additionally, the subroutines require two previously placed query points and a scaling factor.

Algorithm 3: QUERYINTERVAL$([l,r],k)$

Input: The interval $[l,r]$ in which t is known to lie; a number $k \geq 2$ of query points to be placed.
Output: The routine places two query points $q_l < q_r$; then, the search continues in the left or right subinterval or ends.

```
bs ← (r−l)/2, C ← (r−l)/h(k)          ▷ define bisector and scaling factor
q_{l/r} ← bs ∓ g(k)·C                  ▷ place query points q_l, q_r
if k = 3 then
    if t ∈ [l, bs] then
        q_end ← l
    else                                                    ▷ t ∈ [bs, r]
        q_end ← r
else if k > 3 then
    if t ∈ [l, bs] then
        QUERYLEFTSUBINTERVAL([l, bs], q_l, q_r, k−2, C)
    else                                                    ▷ t ∈ [bs, r]
        QUERYRIGHTSUBINTERVAL([bs, r], q_r, q_l, k−2, C)
```

First, consider routine QUERYINTERVAL. Depending on k, the algorithm proceeds as follows: If $k < 2$, we assume that the algorithm aborts without placing any (additional) query point. In all other cases, two query points q_l, q_r are generated and their bisector splits $[l,r]$ in two subintervals $[l,\text{bs}], [\text{bs},r]$ of equal size; q_l, q_r are placed $g(k) \cdot C$ left and right of the bisector where the scaling factor C relates the size of the search interval to the accuracy that we aim for. If $k = 3$, a final query point q_{end} is placed either at l or r to optimally split the subinterval into three parts of equal size; note that this corresponds to an increase in accuracy of $h(3) = 6$. Otherwise, $k > 3$ and more than one query point may still be placed (we already placed two). Hence, the search continues in the subinterval that contains t using the corresponding subroutine. The arguments passed to the subroutine are: the subinterval $[l,\text{bs}]$ or $[\text{bs},r]$, the two query points q_l, q_r that have just been placed, the number of remaining query points and the scaling factor C. Note that q_l, q_r are both at distance $g(k) \cdot C$ from bs; one of which lies inside the subinterval while the other one lies outside. This corresponds to the setting sketched in Figure 2.5a.

Algorithm 4: QUERYRIGHTSUBINTERVAL$([l,r],q_{in},q_{out},k,C)$

Input: The interval $[l,r]$ in which t is known to lie; two query points $q_{out} < l < q_{in} \leq r$ such that $l = \mathrm{bs}(q_{out},q_{in})$; a number $k \geq 1$ of query points to be placed; a scaling factor C.

Output: The routine places a query points q to split the search interval in three parts; then, the search continues recursively depending on the part in which t is found to lie.

$q \leftarrow q_{in} + 2 \cdot h(k-1) \cdot C$ ▷ `place query point` q

$\mathrm{bs}_r \leftarrow (q+q_{in})/2, \quad \mathrm{bs}_l \leftarrow (q+q_{out})/2$

if $t \in [\mathrm{bs}_r, r]$ **then**

  QUERYRIGHTSUBINTERVAL$([\mathrm{bs}_r,r],q,q_{in},k-1,C)$

if $t \in [\mathrm{bs}_l, \mathrm{bs}_r]$ **then**

  $\mathrm{bs}'_l \leftarrow \mathrm{bs}_r - (r - \mathrm{bs}_r)$ ▷ `artificial bisector where` $\mathrm{bs}'_l \leq \mathrm{bs}_l$

  QUERYLEFTSUBINTERVAL$([\mathrm{bs}'_l,\mathrm{bs}_r],q_{in},q,k-1,C)$

else ▷ $t \in [l,\mathrm{bs}_l]$

  QUERYINTERVAL$([l,\mathrm{bs}_l],k-1)$

Finally, we describe how to query a subinterval using QUERYRIGHTSUBINTERVAL for which pseudocode is given in Algorithm 4; QUERYLEFTSUBINTERVAL works symmetrically and pseudocode is given in the Appendix A.3. If $k < 1$, we assume that the subroutine aborts without placing a query point; otherwise, the subroutine places exactly one query point in the subinterval $[l,r]$. A query point q_{in} which lies within the subinterval and another query point q_{out} which lies outside to the left such that l is their common bisector. Hence, by placing a single query point q in the subinterval $[l,r]$, it can be split into three parts; the position of q is chosen $2 \cdot h(k-1) \cdot C$ right of q_{in} corresponding to the setting sketched in Figure 2.5b.[6] These three parts are handled in three cases: In the first case, the target is found to lie in the rightmost part $[\mathrm{bs}_r, r]$; this part can be queried recursively now that q lies within the new search interval and q_{in} lies outside at appropriate distance. In the second case, the target is known to lie within the central part $[\mathrm{bs}_l, \mathrm{bs}_r]$; we increase the size of the this part by shifting bs_l to the left, until its size is exactly as large as the rightmost part[7]. The new left bound of the central interval is denoted with bs'_l, which lies exactly $(r - \mathrm{bs}_r)$ left of bs_r. This modification allows to consider the central part as a left subinterval where q lies outside to the right of the interval at appropriate distance; hence, we may continue recursively with QUERYLEFTSUBINTERVAL, using q_l as the query point in the interval and q as the one outside. In the third and last case, the target hides in part $[l, \mathrm{bs}_l]$ and none of the query points q_{in}, q_{out}, q lies in this subinterval; thus, the overall strategy QUERYINTERVAL is started recursively on the subinterval, ignoring all previously placed query points.

Figure 2.6 exemplifies the incremental search of the algorithm for eight query points.

2.3.4 An Upper Bound for the Accuracy of Incremental Strategies

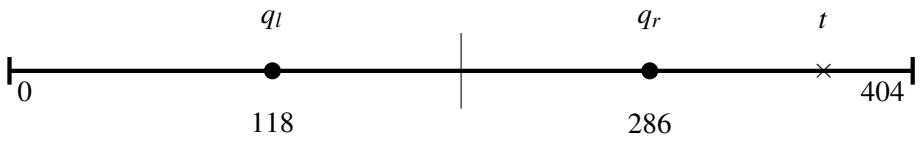
As observed at the beginning of Section 2.3, $(n!)$ is an upper bound for the accuracy of an incremental strategy; in fact, we can prove a much stronger upper bound on the accuracy that can be achieved:

[6] Note that the position of q depends on $h(k-1)$, not $f(k) = h(k-3)$, since two query points have already been placed in the previous calls; again, the scaling factor relates the size of the search interval to the accuracy that we aim for.

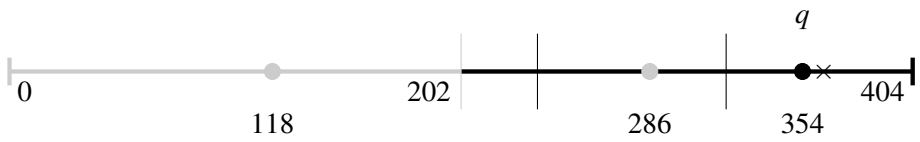
[7] As discussed in Footnote 3 page 24, this is always possible.

$n=$	2	3	4	5	6	7	8
$g(n)=$	-	1	3	7	16	36	84
$h(n)=$	3	6	14	34	76	172	404

(i) Values of the recursive formulas $h(n)$ and $g(n)$.

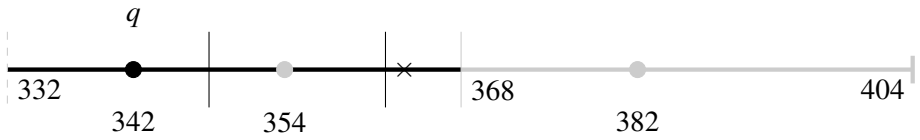


(ii) The initial call of routine QUERYINTERVAL$([0,404],8)$ creates a pair of query points $q_{l/r} = 202 \mp 84$ splitting the interval at 202 in two subintervals of equal size; t is found to lie in the right subinterval.

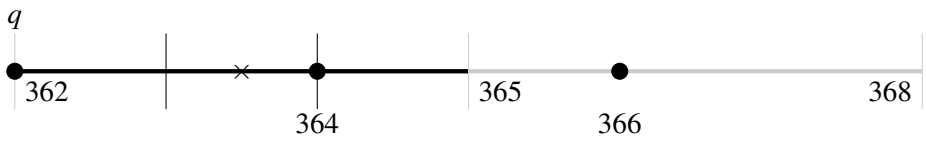


(iii) The call of routine QUERYRIGHTSUBINTERVAL$([202,404],286,118,6,1)$ places a query point at $q = 286+2\cdot 34 = 354$; this generates two more bisectors which split the subinterval in three parts. The target is found to lie in the rightmost part, resulting in another call of QUERYRIGHTSUBINTERVAL.

(iv) The call of QUERYRIGHTSUBINTERVAL$([320,404],354,286,5,1)$ places a query point at $q = 354+2\cdot 14 = 382$. As t is found to lie in the central part, the algorithm creates an artificial bisector $\text{bs}' = 368-36 = 332$.



(v) The call of QUERYLEFTSUBINTERVAL$([332,368],354,382,4,1)$ places q at $354-2\cdot 6 = 342$. This results in a call of QUERYINTERVAL, since t is found to lie in the rightmost part; previously placed points are discarded.



(vi) The call of routine QUERYINTERVAL$([362,368],3)$ starts from scratch and places a pair of points $q_{l/r}$ at 323 ∓ 1; a final query point q_{end} is placed and the target found to lie in the interval $[321,322]$.

Figure 2.6: Target t hides at $363.5 \in [0,404]$. Algorithm 3 pinpoints the target's location to the subinterval $[363,364]$ using 8 queries; binary search would locate t within $[362.96875, 366.125]$.

Theorem 4. *No deterministic, incremental strategy using n query points can locate an arbitrary target in the unit interval with an accuracy* $\Omega(3.66^n)$.

The proof of this theorem is based on the following insight, which we already observed for the optimal query strategy of Observation 3: Previously placed query points need to lie close enough to possibly contribute to accuracy gains when a new point is placed. Suppose, for example, that we have placed $n-1$ query points, and placing another query point q_n generates two new bisectors, which divide the interval containing the target into three equal parts. Thus, the current accuracy improves by a factor three. However, this large increase in accuracy requires two previously placed query points q_i and q_j that satisfy the following two conditions: First, the points q_i and q_j are consecutive in the left-to-right order of all previously placed points; second, the distance between the bisectors $\text{bs}(q_n, q_i)$ and $\text{bs}(q_n, q_j)$ is $1/3$ of the current size of the interval containing the target. Therefore, the distance between q_i and q_j must be $2/3$ of the current size of the interval containing the target.

The previous example illustrates that, for a large increase in accuracy, one needs to have *close pairs*: pairs of previously placed query points with a (comparatively) small distance in between. The example also illustrates that, as the accuracy increases, close pairs can stop being close as their distance relative to the current accuracy increases. Moreover, with the placement of each query point, one can create at most two new close pairs: one with the left neighbour and one with the right neighbour among all query points placed so far.

Definition 3 (accuracy gain). *Let R_n denote the interval in which the target is known to lie after q_n has been placed. Further, let r_n be the size of R_n and a_n the corresponding accuracy $1/r_n$. With $g_n := a_n/a_{n-1} = r_{n-1}/r_n$, we denote the increase of accuracy achieved by placing q_n.*

To create a close pair, one must place a query point q_n close to a previously placed query point, but then the newly created bisectors are also close to the previously created bisectors. In particular, if a close pair is created, most new bisectors inside R_{n-1} will be relatively close to the boundary of R_{n-1} and away from the centre of R_{n-1}. Thus, creating close pairs cannot go together with substantial accuracy gains if the target is near the centre of R_{n-1}. In effect, close pairs are a resource whose scarcity limits the accuracy that can be obtained. We will now make this insight more precise.

Definition 4 (d-close pair). *A d-close pair is a pair of previously placed query points that are consecutive in their left-to-right order and have a distance strictly less than d divided by the current accuracy. Further, let $p_n(d)$ denote the number of d-close pairs after q_n has been placed.*

The following three lemmas capture the trade-off between on the one hand, creating close pairs, and on the other hand, using and losing them while pin-pointing the location of the target. Each lemma assumes that query points $q_1, ..., q_n$ have been placed in response to learning $R_0, ..., R_{n-1}$, which we now consider to be fixed. Within this context, R_n is still a variable: the bisectors of q_n with the previously placed query points subdivide R_{n-1} into a set $\mathscr{R}$ of subintervals, and $R_n \in \mathscr{R}$ is the subinterval among them that contains t. In turn, g_n and $p_n(d)$ are functions of R_n. Each lemma assumes a certain accuracy gain g_n and derives a bound for the net change in the number of d-close pairs $p_n(d)$, for all or for some possible values of R_n. The lemmas address different levels of accuracy gains: *small*, *medium* and *large*. We use the Lemmas 3-5 to prove bounds on the cumulative accuracy gain from successive query points by induction in Lemma 6, which in turn leads to the proof of Theorem 4.

Lemma 3 (small accuracy gain). *If $g_n \geq 1 + d/2$ for all possible $R_n \in \mathscr{R}$, then $p_n(d) \leq p_{n-1}(d) + 1$ for all possible $R_n \in \mathscr{R}$.*

Proof (by contradiction). Suppose two new d-close pairs are created: one with a previously placed point q_l to the left and one with a previously placed point q_r to the right of q_n, by definition. For ease of notation, we may choose the origin of our coordinate system such that R_{n-1} is the interval $(0, r_{n-1})$.

As $g_n \geq 1 + d/2$, placing q_n subdivides R_{n-1} into intervals that each have size at most $r_{n-1}/(1+d/2)$. In particular, the leftmost interval must be bounded from the right by a bisector generated by q_n together with a previously placed query point q_j and we get

$$0 < \mathrm{bs}(q_n, q_j) = \frac{q_n + q_j}{2} \leq \frac{r_{n-1}}{1 + d/2}. \tag{2.11}$$

Since (q_n, q_r) is a d-close pair and $r_n \leq r_{n-1}/(1+d/2)$ holds, we have

$$q_r < q_n + d r_n \leq q_n + d \cdot \frac{r_{n-1}}{1 + d/2}. \tag{2.12}$$

Using both equations, we may now conclude that for the *value* $(q_r + q_j)/2$ we have

$$0 < \frac{q_r + q_j}{2} \overset{(2.12)}{<} \frac{1}{2}\left(q_n + d \cdot \frac{r_{n-1}}{1 + d/2} + q_j\right) \overset{(2.11)}{\leq} \frac{r_{n-1}}{1 + d/2} + \frac{d}{2} \cdot \frac{r_{n-1}}{1 + d/2} = r_{n-1}.$$

Since by definition, the interval $R_{n-1} = (0, r_{n-1})$ cannot contain any bisectors of previously placed query points, we cannot have $0 < \mathrm{bs}(q_r, q_j) < r_{n-1}$ if q_r and q_j are different points. Therefore q_r and q_j must be the same point. Although in this case both points do not create a bisector, we may conclude that q_r lies inside R_{n-1}, since the above equation $0 < (q_r + q_j)/2 = q_r < r_{n-1}$ still holds.

By a symmetric argument, q_l must also lie inside R_{n-1}. However, if R_{n-1} contains both q_l and q_r, then R_{n-1} also contains their bisector, which contradicts the definition of R_{n-1}. Note that q_l and q_r can not be the same point as each of them creates a d-close pair with q_n, by assumption.

Therefore, if two d-close pairs are created, there must be possible locations for the target inside R_{n-1} that lead to $r_n > r_{n-1}/(1+d/2)$. This however contradicts the claimed accuracy gain, which is independent of the target's position. □

Now, assume that the accuracy gain is somewhat higher than what we assumed in the previous lemma. In this case no additional d-close pairs can be created at all, as the following lemma shows. Note that it might even be the case that several d-close pairs are lost.

Lemma 4 (medium accuracy gain). *If $g_n \geq 2 + d/2$ for all possible $R_n \in \mathscr{R}$, then $p_n(d) \leq p_{n-1}(d)$ for all possible $R_n \in \mathscr{R}$.*

Proof (by contradiction). Suppose a new d-close pair is created with a previously placed point q_r. We analyse the case in which q_r lies to the right of q_n, the other case is symmetric. Again, we choose the origin of our coordinate system such that R_{n-1} is the interval $(0, r_{n-1})$, for ease of notation.

Because (q_n, q_r) is a d-close pair, we have

$$q_n > q_r - d r_n \geq q_r - d \cdot \frac{r_{n-1}}{2 + d/2}. \tag{2.13}$$

For each previously placed query point $q_j \neq q_r$, we have

$$\mathrm{bs}(q_n, q_j) = \frac{q_n + q_j}{2} \overset{(2.13)}{>} \frac{1}{2} \cdot \left(q_r - d \cdot \frac{r_{n-1}}{2 + d/2} + q_j\right) = \mathrm{bs}(q_r, q_j) - \frac{d}{2} \cdot \frac{r_{n-1}}{2 + d/2}.$$

Since the bisectors $\mathrm{bs}(q_r, q_j)$ do not lie in R_{n-1}, this implies that the bisectors $\mathrm{bs}(q_n, q_j)$, for $q_j \neq q_r$, cannot lie in the interval $(0, r_{n-1} - (d/2) \cdot r_{n-1}/(2+d/2)] = (0, 2r_{n-1}/(2+d/2)]$. Therefore, the only remaining new bisector, $\mathrm{bs}(q_n, q_r)$, cannot subdivide the interval containing the target, which is larger than $2r_{n-1}/(2+d/2)$, into two pieces of size at most $r_{n-1}/(2+d/2)$.

Thus, if a d-close pair is created, there must be target locations inside R_{n-1} that lead to $r_n > r_{n-1}/(2+d/2)$. This, however, contradicts the claimed accuracy gain, which is independent of the target's position. □

Lemma 5 (large accuracy gain). *For any $c > 1$ and h, d such that $h \geq 2 + d/2$ and $2/h \leq d \leq 2$, the following implication holds:*

If $g_n > h$ for all possible $R_n \in \mathscr{R}$, then there is a possible interval $R_n \in \mathscr{R}$ and a $k > 0$, such that $g_n < c^k (c-1)^{-1} (1 - 2/h)^{-1}$ and $p_n(d) \leq p_{n-1}(d) - k$.

Proof. Suppose that the placement of q_n subdivides R_{n-1} into $m+2$ regions. Note that $m \geq 1$, since otherwise it would not be possible to achieve an increase in accuracy that is strictly greater than $h > 2$. Among these $m+2$ regions, each of size less than r_{n-1}/h, there is a leftmost region and a rightmost region. We call the other m regions *interior* regions and label them $I_1, \ldots, I_m$, in order of decreasing size. Subtracting the size of the leftmost and rightmost region from r_{n-1}, we conclude that the total size of the interior regions is more than $(1 - 2/h) r_{n-1}$. Since $c > 1$ we may also conclude

$$\begin{aligned} \sum_{k=1}^{m} |I_k| &> \left(1 - \frac{2}{h}\right) r_{n-1} > \left(1 - \frac{2}{h}\right) r_{n-1} \cdot \left(1 - \frac{1}{c^m}\right) \\ &= \left(1 - \frac{2}{h}\right) r_{n-1} \cdot (c-1) \cdot \left(\frac{c^{-(m+1)} - 1}{c^{-1} - 1} - 1\right) \\ &= \sum_{k=1}^{m} \frac{(c-1)(1 - 2/h) r_{n-1}}{c^k}, \end{aligned}$$

by using the geometric series. Therefore, there must be a k such that the k-th largest interior region has size more than $c^{-k}(c-1)(1-2/h) r_{n-1}$. If R_n is that k-th largest region, the accuracy gain is upper-bounded by $c^k (c-1)^{-1} (1-2/h)^{-1}$.

It remains to show that in this case, the number of d-close pairs decreases by at least k. Since $h \geq 2 + 2/d$, by Lemma 4, no d-close pairs are created. Each interior region I_i with $i \in \{1, \ldots, k\}$ is bounded by the bisectors of q_n with two previously placed query points q_- and q_+, which were neighbours in the left-to-right order. The distance between q_- and q_+ is

- less than $2r_{n-1}/h \leq d \cdot r_{n-1}$, since, by the conditions of the lemma, I_i has size less than r_{n-1}/h;
- at least $2r_n \geq d \cdot r_n$, since I_i is at least as big as $R_n = I_k$.

Thus, q_- and q_+ constitute a d-close pair before, but not after placing q_n. □

We can now derive the following formula that expresses the cumulative effects of Lemmas 3, 4 and 5 for a complete deterministic query strategy:

Lemma 6. *For any h, d, c, and w such that $h \geq 2 + d/2$ and $2/h \leq d \leq 2$ and $1 < c \leq w$, for each non-negative integer n, it holds that*

$$a_n w^{p_n(d)} \leq b^n$$

for certain locations of the target, where

$$b = \max\left\{\left(1+\frac{d}{2}\right)w^2, \quad \left(2+\frac{d}{2}\right)w, \quad h, \quad \frac{c}{w(c-1)(1-2/h)}\right\}.$$

Proof (by induction). If $n = 0$, we have $a_0 = 1, p_0(d) = 0$ and the claim holds.

Now suppose we add another query point q_n. Distinguishing by the minimum accuracy gain that is achieved for any target in R_{n-1}, one of the following cases must hold:

- Lemma 5 applies. Then, there is a possible $R_n \in \mathscr{R}$ and a k such that $g_n < c^k(c-1)^{-1}(1-2/h)^{-1}$ and $p_n(d) \leq p_{n-1}(d) - k$. Using the induction hypothesis, we conclude

  $$a_n w^{p_n(d)} < \frac{c^k \cdot w^{-k}}{(c-1)(1-2/h)} \cdot a_{n-1} w^{p_{n-1}(d)} \leq \frac{c}{w(c-1)(1-2/h)} \cdot b^{n-1} \leq b^n.$$

  Note that here we use the condition $c \leq w$; without that condition we would not be able to bound $c^k w^{-k}$ to c/w, or to any other constant.

- Lemma 5 does not apply, but Lemma 4 does. So $p_n(d) \leq p_{n-1}(d)$ and, for some $R_n \in \mathscr{R}$, we have $g_n \leq h$, which implies $a_n \leq h \cdot a_{n-1}$. Using the induction hypothesis, we may conclude $a_n w^{p_n(d)} \leq h a_{n-1} w^{p_{n-1}(d)} \leq b \cdot b^{n-1} = b^n$.

- Lemma 4 does not apply, but Lemma 3 applies. So $p_n(d) \leq p_{n-1}(d) + 1$ and, for some $R_n \in \mathscr{R}$, we have $g_n < (2 + d/2)$, which implies $a_n < (2 + d/2) a_{n-1}$. With the induction hypothesis, we get $a_n w^{p_n(d)} < (2 + d/2) w a_{n-1} w^{p_{n-1}(d)} \leq b \cdot b^{n-1} = b^n$.

- None of the Lemmas 3, 4, 5 apply. We have $p_n(d) \leq p_{n-1}(d) + 2$, because each new query point creates at most two d-close pairs, and, for some $R_n \in \mathscr{R}$, we have $g_n < (1 + d/2)$, so $a_n w^{p_n(d)} < (1 + d/2) w^2 a_{n-1} w^{p_{n-1}(d)} \leq b \cdot b^{n-1} = b^n$.

This establishes the induction step, and proves the lemma. □

It remains to pick h, d, c and w such that we get the most out of Lemma 6. Our goal is to prove a strong upper bound, so we want to minimize b. As we can see directly in the definition of b, it cannot hurt to choose d as small as possible and c as large as possible, so we choose $c = w$ and $d = 2/h$ (this also makes the condition $h \geq 2 + d/2$ as permissive as possible). It remains to minimize:

$$\max\left\{\left(1+\frac{1}{h}\right)w^2, \quad \left(2+\frac{1}{h}\right)w, \quad h, \quad \frac{1}{(w-1)(1-2/h)}\right\},$$

subject to $h \geq 2 + 1/h$ and $w > 1$. The maximum of the last three terms is minimized when all are equal: If either one or two of them were higher than the other, it would be possible to adjust h and w slightly such that the higher terms are lowered. Solving for w and h gives us that the last three terms are equal when h is the largest root of $h^3 - 4h^2 + h + 1$, so h is slightly smaller than 3.6511 and $w = h/(2 + 1/h)$ is approximately 1.6057. The first term is of no concern since for these values of h and w, which minimize the other three terms, the first term is even smaller. Thus we get $b = h \approx 3.6511$.

Since $p_n(d)$ cannot be negative, there are targets for which $a_n \leq a_n w^{p_n(d)} \leq b^n$ for all n, and thus, such targets are not located with accuracy better than $\Omega(b^n)$. This concludes the proof of Theorem 4.

2.4 Further Variants and Higher Dimensional Settings

In the following, we first consider an online setting of the incremental variant where the number of query points n is *not* known in advance. Then, we consider modifications of the one-shot variant where t hides on the unit circle, the unit square or unit disk.

2.4.1 The Online Incremental Variant

The incremental variant of our search problem can also be studied in an online setting. In such a setting, the exact number of query points n remains unknown until the last query point has been placed; i. e., whether q_i may be placed or not is only revealed after placing q_{i-1}. The original variant where n is known from the beginning is also called the offline incremental variant in the following. The challenge is to develop strategies with high accuracy that do not depend on the number of query points.

As already for the offline incremental variant, a binary search approach achieves an accuracy of 2^{n-1} in the online incremental variant. The binary search approach uses only the previously placed query point q_{i-1} to decide where to place q_i; thus, the position of q_i does not depend on the overall number of query points. This approach can be even extended to query higher dimensions.

Also in the online incremental variant it is possible to do better than the binary search approach by adapting the incremental query strategy as follows. In the initial routine, the position of the first two query points q_l, q_r can be expressed as $(1/2 \mp g(k)/h(k)) \cdot (r-l)$, cf. Algorithm 3 and Figure 2.5a. Similarly, in the i-th consecutive call of a subroutine, the new query point q lies at $q_{in} + 2 \cdot f(k+1-i)/h(k) \cdot (r-l)$ where q_{in} denotes the point within the current search interval, cf. Algorithm 4 and Figure 2.5b. To make the position of the query points independent of their overall number, we choose $h(n) = b^n$ and $g(n) = 1/4 \cdot h(n) - 1/2 \cdot h(n-3) = (1/4 - 1/2b^3)b^n$. Thus, we can compute the position of q_l, q_r relative to the size of the initial search interval using the ratio

$$R_1 := \frac{g(k)}{h(k)} = \frac{1}{4} - \frac{1}{2b^3} \approx 0.2089$$

in the initial routine; more precisely, we replace the first two lines of Algorithm 3 with $C \leftarrow (r-l)$, $i \leftarrow 0$ and $q_{l/r} \leftarrow (1/2 \mp R_1) \cdot C$. Now, C denotes the size of the search interval on which the initial routine has been started; the new variable i counts the number of consecutive calls of a subroutine and is passed to every subroutine as an additional argument. Similarly, we rewrite the ratio in the i-th consecutive call of a subroutine as

$$\frac{h(k-3+1-i)}{h(k)} = \frac{h(k-3)}{h(k)} \cdot b^{1-i}, \quad \text{define} \quad R_2 := \frac{h(k-3)}{h(k)} = \frac{1}{b^3} \approx 0.0823,$$

and place q in every subroutine relative to the position of q_{in}. More precisely, we replace the first line of Algorithm 4 by $i \leftarrow i+1$ and $q \leftarrow q_{in} + 2 \cdot R_2 \cdot b^{1-i} \cdot C$. Finally, the position of query points are independent of the overall number of query points and we obtain the following result.

Corollary 1. *In the online incremental variant, there is a strategy to locate a target in the unit interval with accuracy $\Omega(b^n)$ where $b > 2.2993$ is the largest real root of $b^4 - b^3 - 6b - 2$.*

2.4.2 The One-Shot Variant on the Unit Circle

As an intermediate step towards two dimensional strategies, one can also study one-shot strategies on the unit circle. In contrast to the unit interval, every bisector intersects with the unit circle at two positions. With an analysis similar to Theorem 1, one obtains a new upper bound of $(n^2-n) \in O(n^2)$ for the accuracy of every one-shot strategy; namely, the number of bisectors is $2\binom{n}{2}$ which generate at most this number of subintervals on the unit circle.

To achieve high accuracy for this variant of the problem, one can adapt the one-shot strategy $\textsc{Gap}_x$ as follows. Instead of two blocks of $(x+1)$ query points at the left and right end of the interval, we start with a single block of $(x+1)$ query points and build gaps in both directions along the boundary of the circle. Applying an analysis similar to Section 2.2.2 we obtain the following corollary.

Corollary 2. *For the one-shot variant on the unit circle, there is a strategy to locate a target with accuracy* $1/2 \cdot (n^2+4n-9) \in \Omega(n^2)$.

Proof. Consider the query points generated by $\textsc{Gap}_x(n)$. Since every pair of points generates a bisector to the left and right on the circle, we may identify the first and the last block of $(x+1)$ query points; more precisely, we match q_i and $q_{\varphi_x(n)-(x+1)+i}$ for $1 \le i \le x+1$. Thus, we obtain a new function $\varphi_x^\circ(n)$ for the number of query points of an equivalent $\textsc{EquiDist}(\varphi^\circ(n))$ strategy on the circle. Counting query points and gaps as before, we get

$$\varphi_x^\circ(n) = ((x+1)+2x)+x+(n-(x+3))\cdot(x+1) = n(x+1)-x^2-2.$$

Choosing $x = \lceil (n-1)/2 \rceil$ optimally, we obtain $\varphi^\circ(n) = 1/4\left(n^2+4n-9\right)$. This completes the proof since the accuracy, equal to the number of subintervals, is exactly $2\cdot\varphi^\circ(n)$ on the unit circle. □

Similar to the one-shot variant on the unit interval, a constant factor gap of $1/2$ between upper and lower bound remains.

2.4.3 One-Shot Strategies in Two Dimensions

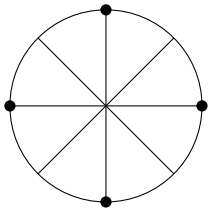
Now let us turn our attention to the two-dimensional case. In the following, we assume that the target hides in the unit square or the unit disk. Applying the techniques presented in Section 2.2, we obtain quadratic lower and upper bounds for the accuracy of one-shot strategies for both types of search spaces.

Corollary 3. *No deterministic strategy using n query points in the unit square (or unit disk) can locate an arbitrary target with accuracy larger than* $O(n^2)$.

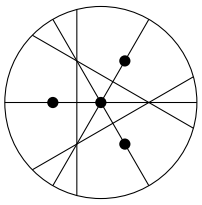
Proof. For the proof, we consider the case where R is the unit square; a similar argument can be applied for the case where R is the unit circle.

As n query points generate at most $1/2 \cdot (n^2-n)$ bisectors, the number of cells in the arrangement $\mathscr{A}$ of bisectors is at most $1/8 \cdot (n^4-2n^3+3n^2-2n+8)$.[8] Let δ denote the maximum diameter of all cells, i. e., $\delta := \max_{C\in\mathscr{A}} \operatorname{diam}(C)$; Since a circle of radius $\delta/2$ maximizes the area that can be covered with diameter δ, the portion of R covered by a cell of diameter δ is not larger than $\pi\cdot\delta^2/4$; this gives an upper bound of the area that all cells can cover together. In turn, since this area is at least 1, this gives a lower bound $\delta \ge 4\sqrt{2}\cdot(\pi(n^4-2n^3+3n^2-2n+8))^{-1/2} \in \Omega(n^{-2})$ on the smallest maximum diameter that can be achieved. Hence the accuracy is $\mathrm{O}(n^2)$. □

[8] Note that an arrangement of k lines in the plane has at most $\binom{n+1}{2}+1$ cells; see, e. g., [44, pp. 4 ff.].



(i) The GAP-strategy extended to the disc where query points are placed only along coordinate axes.



(ii) Could this be an optimal placement of four query points in the one-shot variant on the unit disc?

Figure 2.7: In higher dimensional settings, it might be better to place one of the query points in the center of the disk, instead of placing queries only along coordinate axes.

Note that a slightly smaller constant factor hidden in the big-O notation can be achieved using a tighter upper bound for the maximum area of a single cell, such as $(2\pi/3 - \sqrt{3}) \cdot \delta^2$.

Corollary 4. *In the one-shot variant, a target can be located in the unit square (or unit disk) with accuracy $\Omega(n^2)$ with n query points.*

Proof. Define $k := \lfloor n/2 \rfloor$. As we want to prove a lower bound for the accuracy, we may choose to use only $2k$ query points. We choose our coordinate system such that the center of the unit square (or unit disk) lies at $(0,0)$. Then, we apply $\text{GAP}_x(k)$ along each axis of the coordinate system. Now consider only the bisectors perpendicular to the coordinate axes. These bisectors subdivide the unit square (unit disk) into small rectangles (or parts of small rectangles), each of height and width at most $1/2 \cdot (\varphi_x(k) - 1)^{-1}$, by Lemma 1. Consequently, with $x = \lceil (k-3)/4 \rceil$, the diameter of each square is at most $4\sqrt{2}\left(k^2 + 6k - 27\right)^{-1}$, which gives the quadratic lower bound for the accuracy in terms of n. □

To search a d-dimensional cube or ball, we could place roughly n/d query points on each coordinate axis in a round-robin fashion; on each axis, we place the points according to the one-shot, incremental, or online incremental strategy. Thus, where we obtain accuracy $h(n)$ in one dimension, we can pinpoint a target to a cube of width $h(\lfloor n/d \rfloor)^{-1}$ in d dimensions. This approach, however, fails to take advantage of bisectors between query points placed on different axes, not to mention query points placed more freely. Indeed, for $d = 2$, there are better solutions for $n \in \{3,4\}$, see Figure 2.7. This shows that GAP-strategy is not optimal when extended to the d-dimensional case for $d \geq 2$; yet, it remains unknown whether GAP_x is optimal in the one-dimensional case.

2.5 Conclusion and Outlook

In this chapter, we examined the problem of searching for a stationary target point in a bounded part of the Euclidean space. We considered a new query model where query points must be placed in the search space; the feedback is an ordering of the query points by distance to t. With respect to the feedback, we distinguished two variants: In the one-shot variant all query points had to be placed at once, whereas in the incremental variant points were placed one by one depending on previous feedback.

For the one-shot variant of our search problem, we presented a (trivial) upper bound and developed a lower bound for the accuracy that can be achieved. Only a constant-factor gap between both bounds remains when the search space is a line, a circle, or even a disk or square in the two-dimensional

setting. One approach to narrow the gap might be to improve the trivial upper bound of Theorem 1. Another approach could be to further reduce redundancy among the bisectors generated by GAP; most redundant bisectors are located towards the center and towards the left/right boundary of the interval. Improvements for the one-shot variant on the unit interval immediately imply improvements for the higher dimensional setting and other search spaces, e. g., for the unit circle.

For the incremental variant, we developed a non-trivial upper and lower bound. In this variant, the remaining gap between upper and lower bound grows exponentially in n. Our query strategy does not seem to leave room for substantial improvements; hence, further progress must come from an entirely new query strategy, or from tightening the upper bound. Observe that in each step, our query strategy only uses the bisectors which are formed with the two previously placed query points that are closest to the target. If one can prove that there is an optimal search strategy with this property, this would immediately improve the upper bound to $O(3^n)$.

Apart from the variants discussed in this chapter, other interesting variants can be defined when the target may slightly move between successive query points are placed or when the responses to queries may be inconsistent. In the first case, one can ask questions such as: How much movement of the target would prevent us from locating it? Or, how fast can we locate a target as a function of its speed of movement? Regarding the second case, an example of an inconsistent response could be that, in order of increasing distance from the target, we find the query points $1/4$, $3/4$, and $1/2$. In such cases, the problem may become one of finding the target location that is, in some sense, most consistent with the responses.

Finally, we conclude this chapter comparing the performance of the incremental strategy of Algorithm 3 to that of the binary search approach. As mentioned above, our incremental query strategy outperforms the binary search approach in the new query model. Recall the definition of accuracy as the reciprocal of the initial interval's size divided by the size of the resulting subinterval. To develop a feeling for the exponential decrease of the search interval's size, see Table 2.2. At first sight, the benefit of the incremental strategy compared to the binary search approach seems immense: The difference in the final interval's size grows exponentially with the number of query points. However, this effect is less than one might expect, which becomes apparent when comparing this difference to the size of the initial query interval. As already exemplified in Figure 2.6, the incremental strategy with 8 query points locates t within a final interval of (at most) 1 metre when applied to an initial interval of 404 metres; in contrast, the binary search approach locates t about 3.15 times less precise, i. e., in a final interval of size 3.15 metres. Although this difference keeps growing with the number of query points, it becomes relatively unimportant compared to the size of the initial interval: In the previous example, the difference of roughly $3.15 - 1 = 2.15$ metres corresponds to merely 0.53% of the initial interval's size; for an initial interval of 130 km and 15 query points, it is already less than 0.0064%. In the end, this relative imprecision makes little difference when the number of query points grows.

Nevertheless, the results of this chapter are important for settings where queries are limited or only available at great expenses. The results show that it is possible to achieve the same accuracy using fewer query points as compared to the naive, binary searching approach. For example, instead of placing 18 query points in a binary search fashion, it is possible to achieve the same accuracy using only 15 query points with the incremental strategy, see Table 2.2. Note that saving three of 18 queries cuts the costs by more than 16%. In general, using only $\log_b 2 \approx 83.65\%$ of the query points required by the binary search approach, the incremental strategy achieves the same accuracy. To put it in a nutshell, using resources more efficiently in the new query model, it is possible to save a constant percentage of query points compared to the naive approach.

		approximate size of initial interval / number of query points			
		404 m / 8	130 km / 15	130 km / 18	0.8 ly / 53
approximate size of remaining interval	incremental strategy	1 m	1 m	0.08 m	1 mm
	binary search	3.15 m	8.3 m	1 m	1669 mm

Table 2.2: Comparing the incremental strategy's performance to the performance of the binary search approach. The column specifies the size of the initial interval (in metres, kilometres, or light-years) and the number of query points to place. The maximum size of the final subinterval after all query points have been placed is given in the corresponding row.

Chapter 3

Searching for the Boundary of a Region

This chapter considers the following search problem in a *continuous* search space. A hiker is located at an unknown position in a forest modelled by a simply connected region R in the Euclidean plane. What is a good path for the hiker to follow in order to reach *an arbitrary* boundary point of R within a reasonable amount of time?

If the shape of R is unknown, the hiker's dilemma clearly is: Should one start exploring the area close-by and expand the search radii gradually? Or should one rather pick some direction and run straight on?

We employ a competitive analysis to prove that a specific spiral strategy achieves a reasonable competitive factor for the case where the forest has a non-empty kernel; moreover, if the hiker's unknown starting position lies in the kernel of the forest, this strategy is (almost) optimal w. r. t. the competitive factor. As a basis for our competitive analysis, we introduce a new measure of intrinsic complexity for instances of this escape problem, which we compare to several known shortest escape paths. Finally, we exemplify how the notation and the lower bound technique developed throughout this chapter can be applied to other geometric search problems in order to prove the optimality of given search strategies.

3.1 Introduction and Formal Problem Statement

In 1956, Bellman [11] posed the following research problem: Given a region R and a random point within the region; determine the path which minimizes the maximum time required to reach the boundary. Though it attracted a lot of interest, the problem has only been solved for several special cases to this day. In 2002, Williams [103] even incorporated it into his list of *Million Buck Problems*; a collection of problems where he believes that a solution to any of them will be worth at least one million dollar to Mathematics.

For a formal treatment of the problem, we use the following notation in the style of Finch and Wetzel [36]. A *path* γ is a continuous and rectifiable mapping of $[0,1]$ into $\mathbb{R}^2$. We consider $\gamma(0)$ as the *start* of the path and $\gamma(1)$ as its *end*; with $\{\gamma\}$ we denote the *trace* of γ, i. e., $\{\gamma(t) : 0 \leq t \leq 1\}$, and with $|\gamma|$ we denote its *length*. Similarly, we use the terms for any (sub-)path $\gamma(a,b)$ of γ from $\gamma(a)$ to $\gamma(b)$. A rotation ρ can be applied to (points $\gamma(t)$ of) the path γ, which we denote by $\rho(\gamma(t))$ and $\{\rho \circ \gamma\}$ accordingly. Assume that a simple (not necessarily closed) Jordan curve, i. e., a intersection-free path, divides the Euclidean plane into exactly two regions; the region that contains a specific *starting point* $s \in \mathbb{R}^2$ is denoted by R. We denote the *boundary* of R by ∂R; moreover, with $\ker(R)$ we denote the *kernel* of R, i. e., the set of points $p \in R$ such that all of ∂R is visible from p.

Definition 5 (escape path). *Given a region R and a position $s \in R$. We call a continuous and rectifiable mapping $\gamma : [0,1] \to \mathbb{R}^2$ an* escape path for *s if $\{\rho \circ \gamma\} \cap \partial R \neq \emptyset$ holds for every rotation ρ with $\gamma(0) = \rho(\gamma(0)) = s$. Moreover, we call γ an* escape path for *R, if γ is an escape path for every $s \in R$.*

Using this notation, Bellman's problem is equivalent to finding a shortest escape path for a given region R. Unfortunately, shortest escape paths are known only for special types of regions; for many other simple types, neither the length nor the shape of a shortest escape path is known. Nevertheless, it would be desirable to have at least *some* general escape strategy which behaves almost as good as an unknown shortest escape path for R. In contrast to a shortest escape path, such an escape strategy needs to be less dependent on the size and shape of a specific region; this implies that a search strategy has no specific *end*, which motivates the following definition as a mapping of $\mathbb{R}_{\geq 0}$ into the Euclidean plane.

Definition 6 (escape strategy). *Given a family of regions $\mathscr{R}$ and a position $s \in \mathbb{R}^2$. We call a continuous and rectifiable mapping $\gamma : \mathbb{R}_{\geq 0} \to \mathbb{R}^2$ with $\gamma(0) = s$ an* escape strategy for *$\mathscr{R}$ if, for all $R \in \mathscr{R}$ such that $s \in R$, $\{\rho \circ \gamma\} \cap \partial R \neq \emptyset$ holds for every rotation ρ with $\gamma(0) = \rho(\gamma(0)) = s$.*

For an escape path or escape strategy γ, we write *γ escapes at time $t \in \mathbb{R}$* if and only if t is the first time that $\gamma(t)$ reaches ∂R; i. e., $\gamma(t) \in \partial R$ and $\nexists\, t' < t : \gamma(t') \in \partial R$. As mentioned above, we seek for an escape strategy γ that approximates the length of the shortest escape path with a constant factor for every region R and all $s \in R$.

Definition 7 (approximation of an escape path). *Let $c \in \mathbb{R}^+$, γ be an escape path (or escape strategy respectively) and π_R be an escape path for R. We call γ a* c-approximation of π_R for *R if the following holds.*

For every $s \in R$, there exist $t_s \leq 1$ (or $t_s < \infty$ respectively) and $t'_s \leq 1$ such that

- *γ escapes at time t_s from R in the worst case;*
- *π_R escapes at time t'_s from R in the worst case;*
- *$|\gamma(0,t_s)| \leq c \cdot |\pi_R(0,t'_s)|$ holds.*

Moreover, we call it a c-approximation for a family of regions *$\mathscr{R}$ if γ is a c-approximation of π_R for every $R \in \mathscr{R}$.*

Since shortest escape paths are unknown for most regions, it is hard to compare them directly to certain strategies. For this reason we proceed in two steps: In a first step, we assume that we are given the shape of R and the position of the starting point $s \in R$. Based on this information, we introduce an escape path for s of a special kind which we call the *certificate path*. We provide evidence for why the certificate path may serve as an appropriate substitute for the unknown shortest escape path in general. In a second step, we discuss how to approximate the certificate path when neither information about R nor $s \in R$ is available; without this information, the precise shape or length of both, the shortest escape path and the certificate path, are unknown.

Organisation of this chapter. Section 3.2 presents previous results regarding Bellman's problem and research related to our approach. Section 3.3 introduces and discusses the *certificate path* as a new notion of the intrinsic complexity for instances of the shortest escape path problem. In Section 3.4, we show how the certificate path can be approximated: We provide a spiral strategy that approximates the

certificate path with a factor smaller than 8.1125 for the family of regions with a non-empty kernel; we prove that this strategy approximates the shortest escape path with an optimal factor of 3.3187 for the family of fat convex shapes, equilateral triangles and infinite strips. In Section 3.5, we prove a lower bound for the approximation of the certificate path: For the family $\mathscr{R}$ of regions with a non-empty kernel, there is no c-approximation for $\mathscr{R}$ with a factor $c < 3.3131$; we discuss how this factor can be improved which establishes the optimality result of Section 3.4. Moreover, we exemplify how the notation and the lower bound technique can be applied to another geometric search problem in order to prove optimality of a specific spiral search strategy. Finally, we conclude with Section 3.6.

3.2 Previous Results and Related Work

In this chapter, we apply a technique inspired by Kirkpatrick [64] to develop provably good escape strategies for Bellman's problem. This section gives background information on both: previous results to Bellman's problem and Kirkpatrick's technique. First, we review solutions to special cases of Bellman's problem; then, we give a short introduction to competitive analysis and a short summary of Kirkpatrick's technique.

3.2.1 On Bellman's Problem and Special Cases

Bellman [11] originally stated his research question as follows. *"Given a region R and a random point $s \in R$. Determine the paths which: (i) Minimize the expected time, or (ii) minimize the maximum time required to reach the boundary."* With regard to these two objectives, he was particularly interested in two special cases where R is (a) the region between two parallel lines at known distance d apart and (b) the semi-infinite plane where d is the initial distance to the boundary. In this chapter, with Bellman's problem we only refer to the minimization objective (ii), i. e., finding a shortest escape path for a region R.

Bellman's problem has been given several other names. In 1962, Burr [84, pp. 23-24,149] restated it under the name *river-shore* problem: a swimmer got lost in a river and tries to swim to the shore as quick as possible. In the context of search games [3], the *swimming-in-the-fog* problem is a game of two players: a *searcher*, who tries to reach the boundary of a known shape following a single finite path starting at s; a *hider*, who can rotate and translate the environment so that the path of the searcher will cross the boundary as late as possible. Finally, in their great survey, Finch and Wetzel [36] refer to it as the *lost-in-the-forest* problem: A hiker got lost in a forest of known shape R and wants to follow a shortest escape path for R to leave the forest as quick as possible. Finch and Wetzel also establish the connection with *Moser's worm* problem (see [81], LM 50), which conversely asks for the smallest convex region that covers any path of fixed length.

To date, Bellman's problem has only been solved for some special cases of R. For several special cases, we present shortest escape paths in Section 3.3 and compare them to our certificate path. For the infinite-strip, case (a), Zalgaller [106] found a shortest escape path in 1961; the same solution has been rediscovered and proven by researchers in different fields (e. g., in analysis or computational geometry) [1, 19, 70, 71]. For the half-plane setting, case (b), Isbell [61] found a shortest escape path in 1957. Moreover, shortest escape paths are also known for circular disks, circular sectors, the equilateral triangle, rectangles, fat convex bodies, and some other special cases. All of these results are presented in the survey by Finch and Wetzel [36].

3.2.2 Competitive Analysis and Kirkpatrick's Alternative Cost Measure

Sleator and Tarjan [95] introduced a technique that has come to be known as *competitive analysis*. The technique has been used in many settings since then; see, e. g., [33, 57, 88]. Where classical worst-case analysis classifies two strategies as equivalent w. r. t. their performance, using competitive analysis it is often possible to tell them apart.

An example where competitive analysis has been successfully applied is the m-lane cow-path problem: A single goal is hidden on one of m rays emanating from a common source s; the searcher, initially located at s, is required to find the goal. The cost of the search is given by the distance the searcher covers while exploring the rays and retracing to s. Thus, the cost of any strategy can be arbitrarily increased by placing the goal far away from s; hence, a breadth-first-searching strategy would already be optimal according to a classical worst-case analysis.

Comparing a strategy's cost to the cost of an optimal strategy that knows where the goal lies, differences between strategies are revealed. Baeza-Yates et al. [8] prove that the cost of their strategy exceeds an optimal strategy's cost by a factor of at most $(2em+1)$, where e denotes the natural logarithmic base. This *competitive factor* is also proven to be optimal. Although, at first sight, the m-lane cow-path problem appears to be a discrete version of Bellman's problem, there are two main differences: The searcher has to find a single goal, whereas in Bellman's problem it suffices to reach an arbitrary point of ∂R; moreover, the searcher has to retrace to s when switching from one ray to another, which is not the same for Bellman's problem. These are also the main differences to the following problem.

In [64], Kirkpatrick introduces and analyses[1] the *multi-list-traversal problem*: Given a (multi)set $\mathscr{L}$ of m lists of unknown length, find the end of one of the lists with few traversal costs. It is only allowed to traverse a single list at a time; however, for any list, this process can be interrupted at any position and resumed later. Costs are charged for traversing a list; no costs are charged for switching between lists.

A standard competitive analysis has weaknesses when applied to this problem: The analysis compares an algorithm to a fully informed algorithm which knows the length of each list and only traverses the shortest list to its end. Thus, similar to the cow-path problem, a linear dependency of the competitive factor in m is the best one could hope for; in this sense, a simple breadth-first-searching strategy would already be optimal, although, it does not seem to be a good choice for every instance; see Figure 3.1. Hence, a fully informed strategy does not seem like a suitable choice for the competitor. Moreover, one can argue that the intrinsic complexity of an instance of the problem does not only depend on the length of a shortest list in $\mathscr{L}$ but on the number of lists, as well.

To overcome these problems, Kirkpatrick proceeds in two steps. In a first step, he introduces a different competitor: a partially informed strategy which knows the length of all lists, but not the order in which they are given in $\mathscr{L}$. Thus, he defines a different measure for the intrinsic complexity for a multi-list-traversal problem instance $\mathscr{L}$: The maximum-traversal-cost of a strategy is the maximum cost that the strategy requires for any ordering of the lists in $\mathscr{L}$; the intrinsic maximum-traversal-cost $\xi(\mathscr{L})$ is the minimal maximum-traversal-cost over all deterministic strategies. In other words, the intrinsic complexity of an instance $\mathscr{L}$ is the best that a partially informed algorithm can achieve on all permutations of $\mathscr{L}$. He proves that $\xi(\mathscr{L}) = i^* \cdot |\, l_{i^*} \,|$, where $i^* := \arg\min_{1\le i\le m} i \cdot |\, l_i \,|$ and l_i denotes the i-th longest list in $\mathscr{L}$. Thus, the intrinsic complexity of an instance indirectly incorporates the number of lists. Provably, every deterministic strategy that works correctly on all permutations of $\mathscr{L}$ requires at least cost $\xi(\mathscr{L})$; a strategy that traverses each list up to length $|\, l_{i^*} \,|$, has at most cost

[1] Kirkpatrick [64] also analyses randomized strategies and proves performance guarantees; we only summarize his results for deterministic strategies.

$\xi(\mathscr{L})$, see Figure 3.1 (iii). In a second step, Kirkpatrick introduces a hybrid strategy and proves a tight bound of $O(\xi(\mathscr{L}) \cdot \log(\min(m, \xi(\mathscr{L}))))$ for its traversal cost. Now, the competitive factor is logarithmic in the number of lists m, which highlights the hybrid strategy's strength compared to the standard breadth-first-searching approach.

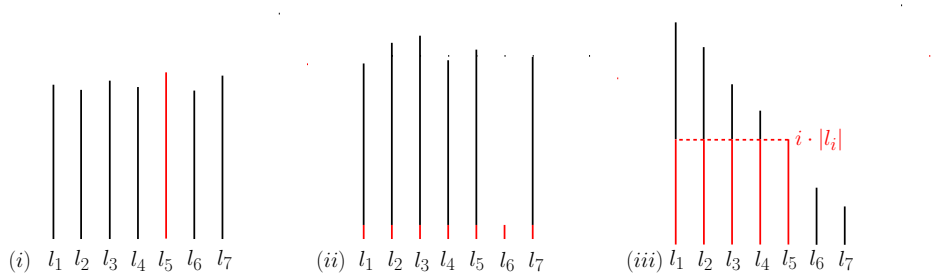


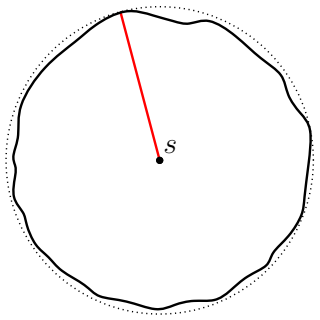
Figure 3.1: Searching for the end of one out of $m = 7$ lists given in a random order. (i) If all lists have about the same length, it is reasonable to pick an arbitrary list and traverse it up to its end. (ii) If one list is significantly shorter, it is reasonable to traverse all lists up to the length of the shortest. (iii) In general, if l_i is the i-th longest list, at most i lists have to be traversed up to length $|l_i|$ to find the end of one of them.

All in all, the use of alternative comparison measures has some tradition. Consider the problem of searching for a point in a polygon and competing against the shortest path. As there is no competitive strategy, other comparison measures have been suggested in this case; see, e. g., [38, 72].

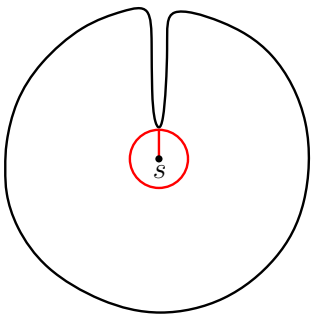
3.3 The Certificate Path

Every escape strategy is trapped in the dilemma of searching the area close-by versus going straight into one direction. While a breadth-first-search approach (BFS) will always result in an escape strategy when expanding the search-radii gradually, this is not true for a pure depth-first-search approach (DFS): the trace is a ray, which can be rotated such that the intersection with ∂R is empty for the case where R is the half-plane. Balancing BFS and DFS in a hybrid strategy which expands the BFS search-radii exponentially, one obtains an escape strategy that approximates the shortest distance d from s to ∂R with a constant factor; the smallest competitive factor is about 27, which we show in Section 3.4.1. In the sense that a constant factor is the best we could hope for, this hybrid strategy is already optimal; however, for certain types of regions, there is an escape strategy that performs much better and for some starting positions we can even prove that it is optimal! To prove this, the shortest distance d is not a constructive choice for the competitor. This is why we replace it by a new notion of complexity which we introduce in the following.

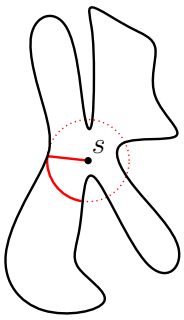
The certificate path is a hybrid strategy that balances BFS and DFS in an optimal way. Assume we have to design an escape path for s in R and we are given the distance from s to ∂R into every direction but not the initial orientation within R. Depending on the specific distances, a pure BFS or pure DFS approach gives an escape path of reasonable length; see Figure 3.2i and ii. For the general case, we define the certificate path as an optimal way of balancing BFS and DFS based on the given distances from s to ∂R.



(i) The distance from s to ∂R is about the same in every direction; hence $\alpha_x \approx 0$.



(ii) The distance x from s to ∂R is very short for only few directions; hence $\alpha_x \approx 2\pi$.



(iii) The distances from s to ∂R vary such that $\alpha_x \approx 1/2 \cdot \pi$.

Figure 3.2: The certificate path π_s depends on the distance from s to ∂R. In the two extreme cases, π_s consists (i) either only of a straight line segment or (ii) the arc segment covers a full circle. In general, π_s balances both extremes; e. g., see (iii).

Definition 8 (certificate path). *Given a region R and $s \in R$. A* certificate path for x and s, *denoted by $\pi_s(x)$, is an escape path that consists of a straight line segment of length x attached to a circular line segment of length $x \cdot \alpha_x$ along the circle of radius x around s.*

A certificate path, *denoted by π_s, is a shortest one among all certificate paths for values of x.*

Note that such a certificate path always exists. The exact values for x and α_x depend on s and R: Intersecting the circle of radius x around s with ∂R, the largest arc segment that lies in R defines $x \cdot \alpha_x$; see Figure 3.2iii. In this restricted family of paths, the certificate path π_s is a shortest escape path.

The definition of the certificate path bears strong resemblance to Kirkpatrick's alternative cost measure for the multi-list-traversal problem introduced in Section 3.2.2. Recall the definition of the intrinsic maximum-traversal-cost $\xi(\mathscr{L})$ as the min-max cost that any deterministic algorithm can achieve over all permutation of the lists $\mathscr{L}$ of given lengths. The certificate path seems like a natural extension of this definition to the two-dimensional setting. This impression basically comes from three common features: First, both cost measures hinge on additional information about the distances to the targets; in the traversal problem this is the length of each list whereas it is the distances from s to ∂R for our problem. Second, they are similar in the sense that both guarantee success as soon as a certain search depth is covered for a certain number of lists or directions respectively. Finally, the cost grows linearly with the search depth for both cost measures. However, there are also three crucial differences to mention: First, $\xi(\mathscr{L})$ depends on the number of lists m whereas π_s does not depend on a limited number of directions; not even if the distance from s to ∂R is given only for a limited number of directions. Second, $\xi(\mathscr{L})$ depends on the depth up to which the lists are traversed; this approaches the *area* of a sector S that is covered during the search, when m grows to infinity. In contrast, the cost of π_s is given by its length $|\pi_s|$ which corresponds rather to the *boundary* of S. Last but not least, $\xi(\mathscr{L})$ is invariant to permutations of the lists which is necessary to prove $\xi(\mathscr{L}) = \min_{1 \le i \le m} i \cdot |l_i|$; however, we require π_s to be only robust under rotations which corresponds to rotations of the ordering of the lists, where the first list is repeatedly appended at the end. When the ordering of the lists is only subject to such rotations, the equality relaxes to $\xi(\mathscr{L}) \le \min_{1 \le i \le m} i \cdot |l_i|$. Similarly, $|\pi_s|$ is only an upper bound for the length of a shortest escape path that can be designed given the distances from s to ∂R. Nevertheless, we are convinced that π_s is a good measure for the inherent complexity of an instance of the shortest escape path problem.

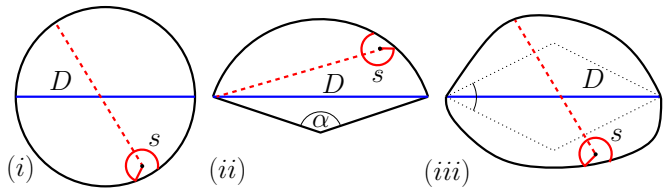


Figure 3.3: Three environments for which the diameter D is the shortest escape path: (i) Circular disks, (ii) circular sectors with $\alpha \geq 60°$, and (iii) *fat* convex sets.

Further, we are convinced that the certificate path π_s can serve as a substitute for the unknown shortest escape path. More precisely, we support the following two claims: (a) For every R and every $s \in R$, π_s approximates the shortest escape path with a small, constant factor; (b) π_s is almost optimal when given the distance to ∂R into every direction, i. e., with a small constant factor, π_s approximates the shortest escape path that is specifically designed given the radial distances from s to ∂R. Intuitively, (a) should hold since a shortest escape path for R is required to be an escape path for every $s \in R$, while π_s is specifically designed for a single $s \in R$ based on the given distances from s to ∂R. To support claim (a), we prove for several cases where a shortest escape path is known that π_s does not escape later than the shortest escape path in the worst case; i. e. π_s approximates the shortest escape path with a factor $c \leq 1$. To support claim (b), we prove that π_s approximates Isbell's path γ_s with a factor of $c < 1.0935$ in the worst case, where γ_s is a shortest escape path when R is the half-plane and the distance from s to ∂R is given. If the reader is willing to accept both claims, the following subsections can be skipped to see how π_s can be approximated.

Last but not least, the certificate path π_s can be efficiently computed for polygonal regions, as we show in Section 3.3.5. This shows that π_s is not only a theoretical tool in the absence of shortest escape paths but can in fact be computed; even though, this is not necessary since it can be easily approximated with the strategies presented in Section 3.4.

3.3.1 Fat Convex Sets, Circular Disks, Circular Sectors, and Regular n-Gons

For several geometric shapes, the diameter is known to be a shortest escape path. Circular disks, circular sectors with an opening angle of at least 60°, and regular n-gons for $n > 3$ are special cases of a more general family of geometric shapes: *fat, convex sets*. A compact, convex set $S \subset \mathbb{R}^2$ is called[2] *fat* if it contains two points p, q such that: (i) the straight line segment $\overline{pq}$ from p to q is the diameter of S; (ii) a rhombus with smaller inner angle of 60° and $\overline{pq}$ as its smaller diagonal fits in S. For such a rhombus, Poole and Gerriets [86] proved, in the notion of Moser's worm problem, that the diameter is the shortest escape path; hence, the diameter is also the shortest escape path for all fat geometric shapes.

It is not difficult to see that π_s approximates the diameter with a factor $c \leq 1$. The definition of π_s allows α_x to be equal to zero; in this case, x must be equal to the largest distance from s to ∂R. If, for example, s lies at the center of a disc of radius D, π_s consists of a straight line segment of length $D/2$, see Figure 3.2i; hence, π_s does not escape earlier than the straight line shortest escape path. For any other position of s, π_s is not longer than the maximum distance from s to ∂R due to its definition, see Figure 3.3; thus it escapes not later than the shortest escape path in the worst case. Altogether, for any $s \in R$, π_s approximates the diameter with a factor $c \leq 1$.

[2]Note that there are several other definitions of fatness.

3.3.2 Besicovitch's Escape Path for Equilateral Triangles

To prove that the certificate path π_s outperforms Besicovitch's [12, 24] shortest escape path γ for the equilateral triangle, we proceed as follows. Let T be an equilateral triangle with sides of unit length and let t_1, t_2, t_3 be the segments that constitute ∂T. As Figure 3.4i shows, γ is symmetric and consists of three segments of equal length, which constitute a total length[3] of approximately 0.981918. We subdivide possible choices for $s \in T$ into two subsets: (a) Points $s \in T$ for which the distance to t_i is greater than $d := 2x\cos(\alpha) \cdot \sin(\alpha) = 9/28\sqrt{7}$ for at least two segments t_i; (b) all remaining points, for which the distance to t_i is at most d for two segments t_i. For each set, we give an upper bound for $|\pi_s|$ and a lower bound for $|\gamma(0,t)|$, where γ escapes at t in the worst case.

For case (a), we first observe that a maximum distance from s to ∂T is at least $1/\sqrt{3}$, where the min-max is attained when s lies at the center of T. Let C be a corner of T such that: (i) the distance from s to the segments t_i, t_j that touch in C is greater than d; (ii) the distance from s to C is maximized, i. e., there is no other corner C' such that (i) holds and the distance from s to C' is greater. Finally, we obtain the lower bound for $|\gamma(0,t)|$ by rotating γ such that it escapes at $C = \gamma(t)$: Due to (i), γ does not escape earlier than t; by (ii), we obtain $|\gamma(0,t)| \geq 1/\sqrt{3} > x$. This lower bound implies that γ escapes along the second or third segment from s; however, by triangle inequality, the straight line segment from s to C is strictly shorter than $|\gamma(0,t)|$ and an upper bound for $|\pi_s|$.

In case (b), s is at most d from two segments; hence, s lies in one of the corners of the triangles, indicated by gray shaded rhombi in Figure 3.4ii. On the one hand, γ can always be rotated around s such that the first two segments of γ fully[4] lie in T; hence $|\gamma(0,t)| > 2x$ in the worst case. On the other hand, $|\pi_s| \leq |\pi_s(d)| \leq d \cdot (1 + 4\pi/3)$, which is strictly smaller than $2x$.

Altogether, this shows that π_s approximates γ with a factor $c < 1$.

3.3.3 Zalgaller's Escape Path for the Infinite Strip of Unit Width

To prove that the certificate path outperforms Zalgaller's [106] shortest escape path γ for the infinite strip S of unit width, we proceed in two steps. Let t denote the time where γ escapes from S in the worst case. In a first step, we give a lower bound for $|\gamma(0,t)|$ independent of $s \in S$. Then, we show that $|\pi_s|$ stays below this bound for every $s \in S$ based on a case distinction.

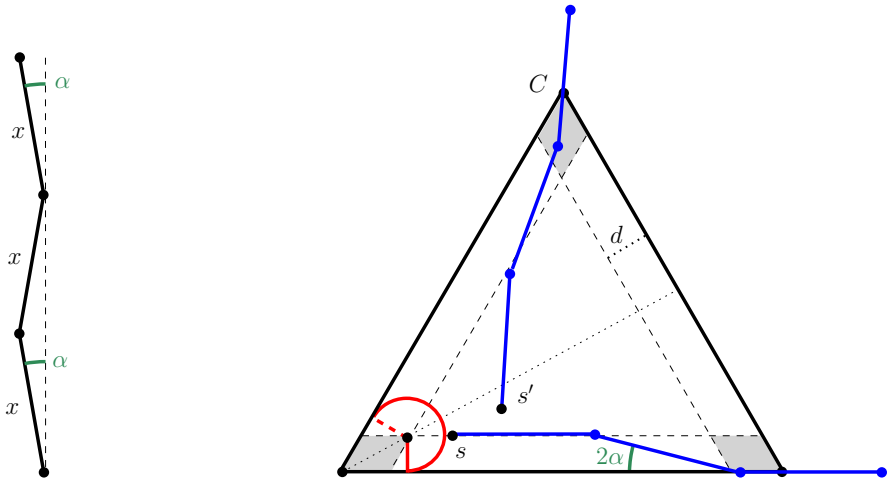
For the first step, consider γ as depicted in Figure 3.5i; if S is the infinite strip of unit width, γ has a total length $|\gamma(0,1)| \approx 2.2783$. For any position s, it is $|\gamma(0,t)| > 2.139$ when escaping at t in the worst-case. This can be seen as follows: W. l. o. g. assume that s is closer to the lower boundary of the strip and γ is rotated such that the top vertex v is arbitrarily close to the upper boundary of the strip; thus, γ escapes through the lower boundary of S at point $v' = \gamma(t)$ and, due to the unit width, the distance from v to v' is arbitrarily close to one. Altogether, when γ escapes at time t, $|\gamma(0,t)| \geq 1/2 \cdot |\gamma(0,1)| + 1 > 2.139$; see Figure 3.5ii.

In the second step, we show that $|\pi_s| < 2.139$ for every $s \in S$ by dividing the points in S into two subsets: (a) points $s \in S$ for which the distance to ∂S is greater than $d := 1/4$; (b) all remaining points, for which the distance to ∂S is at most d. For case (a), consider the certificate path $\pi_s(1)$ for which $\alpha_1(d) = \alpha' + \alpha'' = \arcsin(d) + \arcsin(1-d)$; since $\alpha_1(d) < 1.101$ for $d \in (0.25; 0.75)$, we have $|\pi_s| \leq |\pi_s(1)| < 1 \cdot (1 + 1.101) = 2.101$. For case (b), $|\pi_s| \leq \pi_s(1/4) \leq 1/4 \cdot (1 + 2\pi) < 1.83$.

In both cases, it is $|\pi_s| < 2.139 < |\gamma(0,t)|$, which shows that π_s approximates γ with a factor $c < 1$.

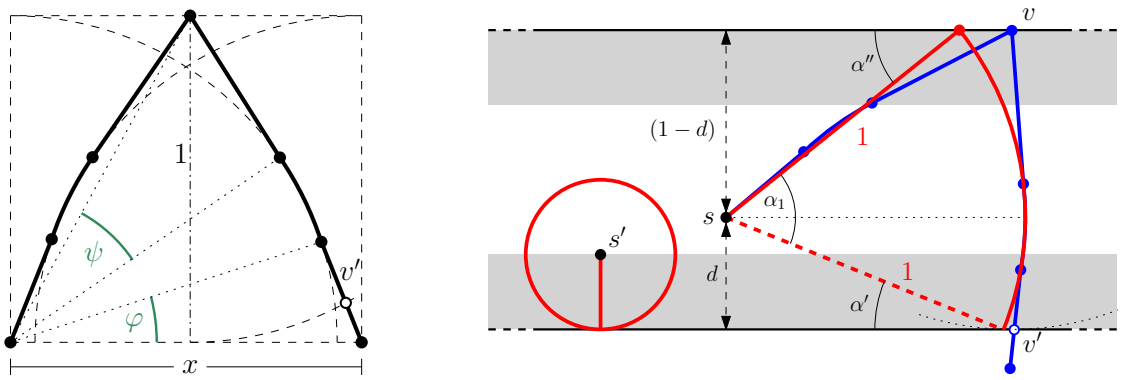
[3] Note that this is shorter than the diameter of the triangle.

[4] The distance from any point in a gray shaded rhombus to the non-adjacent side of the triangle is at least $2x\cos(\alpha)\sin(\alpha)/\cos(\pi/3) > 2x$.



(i) (ii)

Figure 3.4: (i) Besicovitch's escape path is a shortest escape path for the equilateral triangle with unit side length; $\alpha = \arcsin(1/\sqrt{28}) \approx 10.9°$ and $x = \sqrt{3/28}$. (ii) Along dashed lines, the distance to one side of the triangle equals $d := 2x\cos(\alpha)\sin(\alpha) = 9/28\sqrt{7}$. For any point in the shaded rhombi, the distance to two of the sides of the triangle is at most d and greater than $2x$ to the third; the certificate path in the worst case (red) satisfies $|\pi_s| \leq d \cdot (1 + 2/3 \cdot 2\pi)$.



(i) (ii)

Figure 3.5: (i) Zalgaller's escape path in its bounding box of height 1 and width x; the defining values are $\varphi = \arcsin(1/6 + 4/3 \sin(1/3 \arcsin 17/64))$, $\psi = \arctan(1/2 \cdot \sec\phi)$ and $x = \sec\varphi \approx 1.043590$. (ii) Zalgaller's escape path can always be rotated such that vertex v is arbitrarily close to the upper boundary and the path escapes at point $e = \gamma(t)$ via the lower boundary. For points in gray shaded areas, the distance to ∂S is at most $d = 1/4$ and $|\pi_s|$ is bounded by $|\pi_s(1/4)|$; for all other points, $|\pi_s|$ is bounded by $|\pi_s(1)|$.

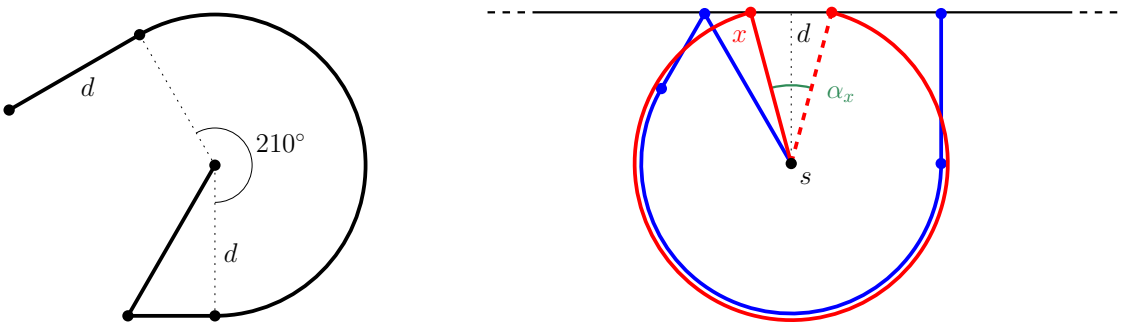


Figure 3.6: (i) Isbell's escape path γ_s where d denotes the distance from s to ∂H. (ii) For $x \geq d$, $\pi_s(x)$ (red) is an escape path with $\alpha_x = 2 \cdot \arccos(d/x)$ that approximates γ_s (blue).

3.3.4 Isbell's Path to Find a Line at Known Distance

Let $\mathscr{H}$ be the family of half-planes. For every $H \in \mathscr{H}$, π_s approximates the shortest escape path for H that is specifically designed given the distance from s to ∂H. Isbell's path γ_s is a shortest escape path when the distance from s to ∂H is given [61]; note that this is equivalent to the information about the radial distances to ∂H around s which are provided to compute π_s. In contrast to the shortest escape paths discussed above, π_s can only approximate γ_s with a factor $c \geq 1$. This is due to the fact that π_s is restricted to a certain shape and only a shortest path within this class of paths; yet, we can prove that π_s approximates γ_s with a small factor.

Lemma 7. *For every infinite half-plane H, π_s approximates Isbell's path with a factor $c < 1.0935$.*

Proof. In [61], Isbell proved that $|\gamma_s| = (\sqrt{3} + 7/6 \cdot \pi + 1)d$ where γ_s denotes the shortest escape path which is specifically designed given the distance d from s to ∂H. W. l. o. g. we may set $d = 1$.

For $x \geq d$, the certificate path for x in H is an escape path and α_x equals $x \cdot (2\pi - 2 \cdot \arccos(1/x))$. Thus, π_s can be obtained by minimizing $x \cdot (1 + 2\pi - 2 \cdot \arccos(1/x))$, which gives $x \approx 1.04356$ and results in an approximation factor smaller than 1.093484. □

This exemplifies that a certificate path, although restricted to a special shape, can approximate optimal solutions based on the same amount of information. Therefore, we are convinced that it is legitimate to focus on the approximation of the certificate path instead of (unknown) shortest escape paths in Section 3.4. In Section 3.4.3, we use the lemma to show how tight our spiral strategy approximates Isbell's path.

3.3.5 Computing the Certificate Path in a Simple Polygon P

In this section we consider the case where P is a simple polygon in the Euclidean plane and describe how to compute the certificate path π_s for a point s in P. Let V denote the visibility polygon of s in P; using subscripts, we distinguish the set of edges E_P of P from the set of edges E_V of V. With a preprocessing step, we ensure that for every $x > 0$, the circle $\partial B_s[x]$ of radius x around s intersects with every edge of E_P or E_V at most once.[5] Let P and V be given by the set of vertices $\{p_1, \ldots, p_n\}$

[5] Every edge of P which does not satisfy this assumption can be split in two edges by inserting a single additional point into ∂P. This preprocessing adds at most n additional points; hence, the complexity of P (and ∂P) remains linear in n.

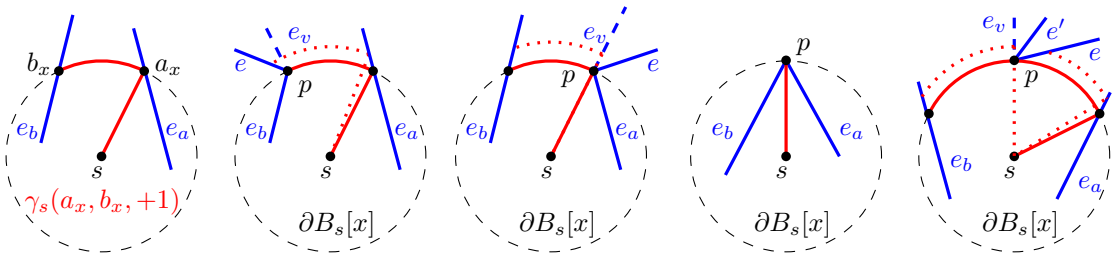


Figure 3.7: Sweeping a polygon with an expanding circle around s; due to the preprocessing step, for fixed radius x, the dashed sweep line $\partial B_s[x]$ intersects with any segment at most once. (i) A candidate defines the red-coloured path $\gamma_s(a_x, b_x, +1)$ for some radius x. At each event, a point $p \in U$ is processed: (ii) p is an endpoint of e_b; (iii) p is an endpoint of e_a; (iv) p is the endpoint of e_a and e_b; or (v) p comes to lie on the circular segment from a_x to b_x.

and $\{v_1, \ldots, v_{n'}\}$ after the preprocessing step. In the following, we denote the union of both set with U; thus, $U := \{p_1, \ldots, p_n\} \cup \{v_1, \ldots, v_{n'}\}$ where $|U| \leq n + n' \in O(n)$.

Apart from the definition above, π_s can also be described using a pair of points $a_x \in \partial V, b_x \in \partial P$ together with a direction of rotation $\sigma \in \{-1, +1\}$ which is either clockwise (-1) or counter-clockwise $(+1)$. Thus, π_s consists of two subpaths: a straight line segment $\overline{sa_x}$ from s to a_x of length x, attached to the σ-oriented circular segment from a_x to b_x along $\partial B_s[x]$. In the following, we abbreviate paths of this shape with $\gamma_s(a_x, b_x, \sigma)$ for short, see also Figure 3.7i. This alternative description motivates the following definition of a *candidate* where a_x and b_x are fixed to specific edges; at the same time, the radius x may vary.

Definition 9 (candidate). *A* candidate *consist of a pair of edges $e_a \in E_V, e_b \in E_P$; a direction of rotation σ; and a maximal existence interval $[x^-, x^+] \subset \mathbb{R}_{>0}$ so that, for all $x \in [x^-, x^+]$, $\gamma_s(a_x, b_x, \sigma)$ lies completely within P where a_x, b_x denote the unique intersections of e_a, e_b with $\partial B_s[x]$.*

The length-function *$\ell : [x^-, x^+] \to \mathbb{R}^+$ of the candidate maps every radius x to the length of $\gamma_s(a_x, b_x, \sigma)$ where a_x, b_x depend on x, i. e., $\ell(x) \mapsto |\gamma_s(a_x, b_x, \sigma)|$. We call $\ell(x)$ the* length *of the candidate at x.*

On the one hand, for every certificate path π_s, a candidate $\gamma_s(a_x, b_x, \sigma)$ exists where $\pi_s(x)$ equals[6] $\gamma_s(a_x, b_x, \sigma)$ for some x in the candidate's existence interval; this allows us to restrict the search for π_s to the set of all candidates. On the other hand, note that there are candidates where $\gamma_s(a_x, b_x, \sigma)$ is not even an escape path for some/every x in their existence intervals. However, for every fixed radius x, a candidate of maximum length at x has to be an escape path. This motivates the following approach.

The idea of the algorithm is to reduce the search for π_s to the set of candidates. Thus, π_s can be obtained by finding a radius x that minimizes the length of a candidate of maximum length at x. For this purpose, the algorithm proceeds in three steps: In a first step, we compute all σ-oriented candidates for each $\sigma \in \{-1, +1\}$; in a second step, for each σ, we compute the upper envelope of the set of length-functions defined by the σ-oriented candidates; finally, we obtain π_s via computing a minimum value of each of the upper envelopes.

On the first sight, there seems to be a quadratic number of candidates: one for every pair of edges. In fact, there is not a candidate for every pair since $\gamma_s(a_x, b_x, \sigma)$ must lie within P. As the following lemma shows, the number of candidates is linear.

[6] up to rotations around s

Lemma 8 (computation of candidates). *The number of candidates is $O(n)$. After $O(n \log n)$ preprocessing time, all candidates can be computed in $O(n)$ deterministic time.*

Proof. The following sweep algorithm computes all candidates by expanding the radius of $B_s[x]$ gradually. In the following, we only consider candidates with counter-clockwise orientation $\sigma = +1$; candidates with $\sigma = -1$ can be computed similarly. Due to the preprocessing step, an event occurs if and only if a boundary point of U comes to lie on $\partial B_s[x]$; thus, we may sort and process the points in U in ascending distance from s. In a sweep status structure, we keep track of all candidates for which the lower bound x^- of their existence interval has been found but the upper bound x^+ is still unknown.

At every event, a boundary point $p \in U$ comes to lie on $\partial B_s[x]$ and at most one candidate $\gamma_s(a_x, b_x, \sigma)$ has to be updated: Two candidates of the same orientation can never overlap for the same value of x; hence, at most one candidate exists where $p \in \gamma_s(a_x, b_x, \sigma)$. In all other cases, $p \notin \gamma_s(a_x, b_x, \sigma)$ and does not affect the maximal existence interval of the candidate. Consequently, no update is required for these candidates. With regard to the affected candidate, we distinguish two cases: (I) Either p is the endpoint of e_a, e_b, or both; or (II) p comes to lie on the circular segment from a_x to b_x.

In case (I), e_a, e_b, or both are completely contained in $B_s[x]$ and will no longer intersect with $\partial B_s[x]$ for radii larger than x; cf. Figure 3.7ii to 3.7iv. Hence, we may set $x^+ := x$ and remove the candidate from the sweep status structure. If p is not an endpoint of both, e_a and e_b, we have to create new candidate. For this purpose, let $e \in E_P, e_v \in E_V$ denote the edges[7] adjacent to p which lie outside of $B_s[x]$.[8] If p is the endpoint of e_a, we create a σ-oriented candidate for the pair of edges (e_v, e_b), cf. Figure 3.7ii; otherwise, p is the endpoint of e_b and we create a σ-oriented candidate for the pair (e_a, e), cf. Figure 3.7iii. In every case, we set $x^- := x$ and insert the candidate into the sweep status structure.

In case (II), we may set $x^+ := x$ and remove the candidate from the sweep status structure since $\gamma_s(a_x, b_x, \sigma)$ does no longer lie completely within P for radii larger than x. As p is neither an endpoint of e_a nor e_b, p is the endpoint of $e, e' \in E_P$ and $e_v \in E_V$[9] which lie outside of $B_s[x]$. The three edges can be ordered in σ-orientation around p; since e_v is an edge of the visibility polygon V, we may assume that it lies to the left or right of the ordering, i. e., w. l. o. g. we assume either the order e, e', e_v or e_v, e', e. In contrast to case (I), we now have to create two new σ-oriented candidates: If the ordering is e, e', e_v, we create two σ-oriented candidates for the pairs of edges (e_a, e) and (e_v, e_b), cf. Figure 3.7v; otherwise, we create two σ-oriented candidates for the pairs of edges (e_a, e') and (e, e_b). For both candidates, we set $x^- := x$ and insert them into the sweep status structure.

We initialise the sweep status structure as follows. Let p be a point of U that lies closest to s, at distance m. Thus, for every $x < m$, $B_s[x] \cap \partial P = \emptyset$ and no candidate exists for radii smaller than m. We create the initial σ-oriented candidate similar to case (II). Let e, e', e_v or e_V, e', e be the ordering of the edges adjacent to p which lie outside of $B_s[x]$: If the ordering is e, e', e_v, create a single, σ-oriented candidate for the pair (e_v, e); otherwise, create a single, σ-oriented candidate for the pair (e, e'). For both candidates, set $x^- := x$ and insert them into the sweep status structure.

The number of events is given by $|U|$, which is linear in n as stated above. After the initialisation step, only a single candidate path exists; moreover, at each event, only a single candidate is updated which implies that the number of new candidates increases by at most two. Hence, the overall number of candidates is bounded by the number of events and linear in n.

[7] Note that $e = e_V$ if e is visible from s.

[8] Note that e could as well lie within $B_s[x]$; however, this case needs to be handled similar to case (II).

[9] Note that $e = e_V$ or $e' = e_V$ if e and e' are visible from s.

After $O(n \log n)$ preprocessing time, each event can be processed in constant time. More precisely, the events of type (I) can be processed in constant time since one can always access the affected candidate via e_a or e_b of which p is an endpoint. However, this is impossible for the events of type (II) as there is no candidate path with segments adjacent to p. Nevertheless, for every point p of U, one can store a pointer to the segments of P and V which $\partial B_s[x]$ intersects before and after p. With this pointer we can find the affected candidate via the segment e_a in constant time. A data structure storing all pointers requires $O(n)$ space and can be computed in $O(n \log n)$ time. As each candidate can be easily accessed via the edges e_a, e_b, a simple list suffices to implement the sweep status structure.

As soon as the last event has been processed, P and V are completely contained in $B_s[x]$; hence, the algorithm terminates. Furthermore, no σ-oriented candidates other than the ones computed during the algorithm can exist: Due to the maximality of the existence interval of every candidate, either a_{x^-} or b_{x^-} is an endpoint of a segment; the endpoint, however, defines an event where the desired candidate is created. □

In a second step, we compute the upper envelope of the length-functions of all σ-oriented candidates as follows. For each set of candidates, we compute the upper envelope of the length-functions, since the candidate of maximum length at x defines an escape path for x. Both sets contain at most $O(n)$ candidates; the length-functions of two candidates intersect at most twice as described below. Thus, since length-functions are only defined on existence intervals, the complexity of their upper envelope is $O(\lambda_4(n))$ where $\lambda_4(n)$ denotes the maximum length of a Davenport-Schinzel sequence of order $2+2=4$ including n distinct values.[10] The upper envelope of the length-functions can be computed in $O(\lambda_4(n) \log n)$ time, cf. [92].

Finally, we obtain the certificate path from both upper envelopes as follows. Sweep over each upper envelope to find a radius for which the upper envelope attains a minimum value. Since the minimum corresponds to a candidate and defines an escape path, the shorter one defines π_s.

Theorem 5. *Given a polygon P with n edges in the Euclidean plane. For every $s \in P$, the certificate path $\pi_s(x)$ can be computed in $O(\lambda_4(n) \log n)$ time where $\lambda_4(n)$ denotes the maximum length of a Davenport-Schinzel sequence of order* 4.

It remains to show that two length-functions can intersect at most twice. The length function ℓ of a candidate is defined for every x within the candidate's existence interval $[x^-; x^+]$ and can be expressed as follows. Let d_a denote the shortest distance from s to the line l_a defined by the edge e_a and let A be the point on this line at distance d_a from s; similarly, we define d_b and B with respect to the edge e_b, cf. Figure 3.8. Thus, we can express ℓ as

$$\begin{aligned} \ell(x) &= (1+\alpha_x) \cdot x \qquad \text{where} \\ \alpha_x &= \measuredangle AsB - \arccos\left(\frac{d_a}{x}\right) - \arccos\left(\frac{d_b}{x}\right). \end{aligned}$$

For two candidates with length functions ℓ and ℓ', consider their difference in length, i. e., $\ell(x) - \ell'(x)$; this difference equals

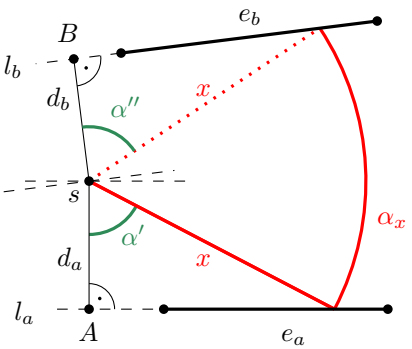


Figure 3.8: The value of α_x can be obtained subtracting α' and α'' from $\measuredangle AsB$.

[10] Note that, for any $t \in \mathbb{N}$, $\lambda_t(n) \in O(n \log^* n)$ where $\log^* n$ denotes the iterated logarithm, i. e., the smallest value m such that $\log(\log(\cdots \log(n))) \leq 1$. Thus, the growth of $\lambda_t(n)$ is practically considered as linear, e. g., $\log^*(2^{65536}) = 5$.

zero if and only if

$$\begin{aligned} f(x) &:= c + \arccos\left(\frac{d_a}{x}\right) + \arccos\left(\frac{d_b}{x}\right) - \arccos\left(\frac{d'_a}{x}\right) - \arccos\left(\frac{d'_b}{x}\right) \\ &= c + \operatorname{arcsec}(d_a x) + \operatorname{arcsec}(d_b x) - \operatorname{arcsec}\left(d'_a x\right) - \operatorname{arcsec}\left(d'_b x\right) \qquad = 0 \end{aligned}$$

where c is a constant that depends on the position of the line segments of both candidates. Since d_a, d_b, d'_a, d'_b are all non-negative, also the sums $(\operatorname{arcsec}(d_a x) + \operatorname{arcsec}(d_b x))$ and $(\operatorname{arcsec}(d'_a x) + \operatorname{arcsec}(d'_b x))$ are strictly increasing with x and can have at most two proper intersection.

Note that all three steps of the algorithm can also be performed in one sweep. However, for the sake of presentation, we decided to explain all three steps separately. Moreover, combining them into a single step does not improve the asymptotic runtime bound: The complexity of the upper envelope remains $\lambda_4(n)$ and the log-factor is still required for the initial sorting of the points.

Further note that the algorithm can be adapted to compute π_s for arbitrary regions R in the Euclidean plane where ∂R is defined using polygonal chains; this includes unbounded regions as well as regions with polygonal holes.

Finally, note that we presumed the underlying computational model to be a real RAM. In the runtime analysis of this algorithm, this is particularly necessary in order to compute the intersections of length-functions. In practice, these intersections can be approximated numerically.

3.4 Approximation of the Certificate Path

In this section, we consider two families of regions and analyse two types of escape strategies to approximate the certificate path π_s. For the family of simply connected regions, an escape strategy can always be obtained by searching along expanding concentric circles. We show that such a strategy can approximate π_s with a factor of roughly 27. For the (sub-)family of regions with a non-empty kernel, we consider spiral searching strategies. We first develop a spiral strategy that approximates π_s with an optimal factor for the case where $s \in \ker(R)$; then, we prove that this strategy approximates π_s with a factor of roughly 8.1125 for arbitrary starting positions $s \in R$.

3.4.1 Searching via Expanding, Concentric Circles in Simply Connected Regions

The following notation is used to describe strategies of expanding concentric circles. Let the coordinate system be chosen such that $s = (0,0)$. Every escape strategy ζ of expanding concentric circles can be described by alternating the following two steps: (1) Follow the horizontal axis to the right for some distance; (2) at the current position $(x,0)$, follow the circular arc segment of radius x around s in (counter-)clockwise direction for 360°. Thus, ζ can be uniquely described by a positive, infinite sequence $X := \{x_i\}_{i \geq 0}$, the elements of which define the radii of the concentric circles, see also Figure 3.9i.

In a preliminary step, we consider how ζ approximates the shortest distance d_R from s to ∂R. W. l. o. g.[11] we may assume that ζ reaches ∂R during step (2) of the $(i+1)$-th iteration; thus, ζ escapes after searching an angular portion $(\alpha \cdot x_{i+1})$ of the circle of radius x_{i+1} around s. Since ζ did not escape in previous iterations, we know that $d_R = x_i + \varepsilon$ for some $\varepsilon > 0$. Thus, we can bound the

[11] Otherwise the competitive-ratio can be increased by creating a new simply connected region R' with the following properties: (i) the shortest distance from s to ∂R is unchanged, i. e., $d_R = d_{R'}$; (ii) ζ reaches ∂R' in the $(i+1)$-th iteration. Figure 3.9i gives a sketch of such a region.

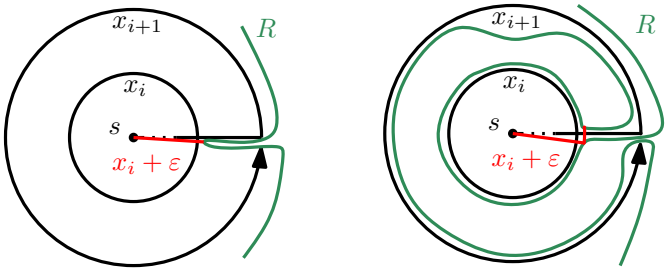


Figure 3.9: An escape strategy ζ of expanding, concentric circles around s. The radii of the circles define a positive sequence $\{x_i\}_{i\geq 0}$. A specific region R is indicated with a green line. Moreover, in (i) the red line segment indicates the shortest distance from s to ∂R; whereas in (ii) the red line indicates the certificate path π_s.

competitive ratio of ζ from above:

$$\frac{x_{i+1}(1+\alpha)+\sum_{j=0}^{i} 2\pi x_j}{d_R} \leq \frac{x_{i+1}+\sum_{j=0}^{i+1} 2\pi x_j}{x_i+\varepsilon} \leq \frac{x_{i+1}+\sum_{j=0}^{i+1} 2\pi x_j}{x_i},$$

where $\alpha \in [0, 2\pi]$. In the worst-case, the competitive ratio comes arbitrarily close to this upper bound for certain regions as, for example, indicated in Figure 3.9i.

To obtain a strategy ζ with a minimal competitive factor, we employ a technique of Gal [3]. For this purpose, we consider the following sequence of functionals

$$F_l(X) = \frac{x_1+\sum_{j=-l}^{1} 2\pi x_j}{x_0} \quad \text{with} \quad F(X) = \frac{x_1+\sum_{j=-\infty}^{1} 2\pi x_j}{x_0},$$

which is motivated by the above considerations. We convince ourselves that these functionals satisfy all necessary preconditions [3, (7.5)-(7.9), p. 110] of Gal's minimax theorem [3, Theorem 7.9]; for a similar functional this is verified in the Appendix B.1. Hence, we may apply the theorem to obtain that a minimal value of the functional is attained by geometric sequences A_a, i. e.,

$$\begin{aligned} \limsup_{i\to\infty} F(X^{+i}) &\geq \inf_{0\leq a\leq\infty} F(A_a) = \inf_{a>1} \frac{a^1+\sum_{j=-\infty}^{1} 2\pi a^j}{a^0} = \inf_{a>1} a^1 + 2\pi \sum_{j=-1}^{\infty} a^{-j} \\ &= \inf_{a>1} \frac{a^2(2\pi+1)-a}{a-1} = 1+4\pi+2\sqrt{2\pi(1+2\pi)} \approx 27.096, \end{aligned}$$

where $a = 1+\sqrt{2\pi/(1+2\pi)} \approx 1.9288$. Thus, among all strategies of expanding, concentric circles, the strategy described by the sequence $\{a^i\}_{i\geq 0}$ achieves a smallest competitive factor.

Now, we turn the attention to the approximation of the certificate path. The family of simply connected regions $\mathscr{R}$ contains also regions for which the certificate path becomes arbitrarily close to d_R, the shortest distance from s to ∂R; an example is indicated in Figure 3.9ii. For these types of regions, the certificate path consists of a line segment of length $(x_i+\varepsilon)$ attached to a circular segment of length $(\varepsilon \cdot x_i)$. Moreover, $|\pi_s|$ approaches d_R since ε can be chosen arbitrarily small. Hence, when approximating the certificate path, ζ can not achieve a better approximation factor as when approximating d_R and we obtain the following theorem.

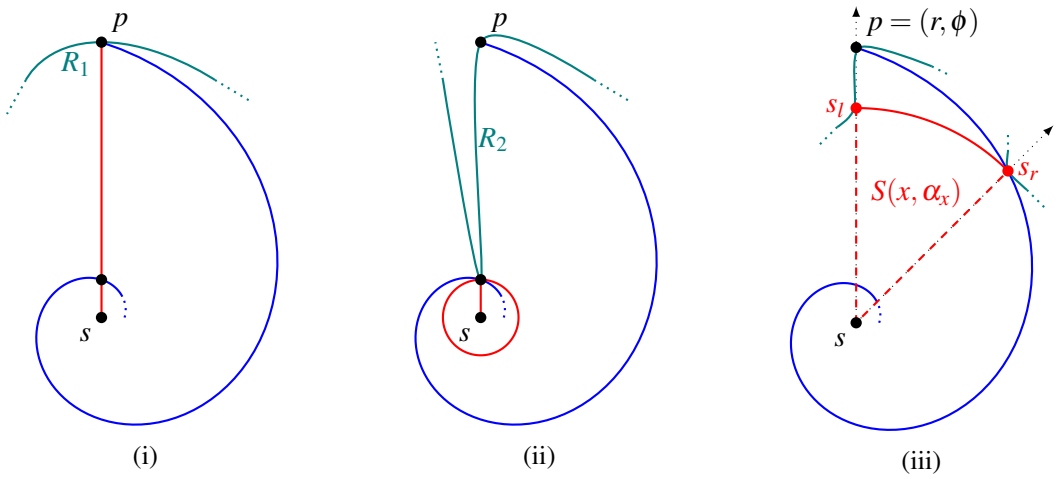


Figure 3.10: Comparing the spiral path ζ (blue) to the certificate path π_s (red). In the two extreme cases, π_s is either a straight line segment (i) or covers a full circle around s (ii). In the general case (iii), π_s covers a circular sector $S(x,\alpha_x)$.

Theorem 6. *For the family of simply connected regions $\mathscr{R}$, the escape strategy of expanding, concentric circles approximates the shortest distance to ∂R (and the certificate path for s) with a factor of* 27.096. *The radius in the i-th iteration is given by a^i where $a = 1+\sqrt{2\pi/(1+2\pi)}$.*

In the sense that a constant approximation factor is the best that we could hope for, Theorem 6 gives already a sufficient solution; however, the approximation factor is far from the lower bound which we establish in Section 3.5. For this reason, we consider a special type of simply connected regions in the subsequent section: regions with a non-empty kernel. We develop a strategy which achieves a much better approximation factor for this subclass of regions.

3.4.2 Spiral Search in Regions with a Non-Empty Kernel

The approach presented in Section 3.4.1 can also be applied to the family of regions with a non-empty kernel to develop a strategy of expanding, concentric circles with a smallest approximation factor. Such a strategy achieves an approximation factor of roughly 8.14; however, this is far from the lower bound, which we establish in Section 3.5. Hence, in the following, we consider spiral strategies instead. We develop a strategy which achieves an optimal approximation factor for the case where the starting position s lies in the kernel $s \in \ker(R)$ of the region R. Let π_s denote the certificate path; its length is given by $|\pi_s| = x\cdot(1+\alpha_x)$ for some $x \geq 1$ and $\alpha_x \in [0,2\pi]$.

A logarithmic spiral Γ can be defined using polar coordinates $(a\cdot e^{\varphi\cot(\beta)},\varphi)$; the angular coordinate is given by $\varphi \in (-\infty,\infty)$ and the radial coordinate is expressed by a constant a and eccentricity β. The Euclidean distance from s to a coordinate $(a\cdot e^{\phi\cot(\beta)},\phi)$ is given by the radial coordinate; the arc length $|\Gamma(-\infty,\phi)|$ is equal to $a/\cos\beta\cdot e^{\phi\cot(\beta)}$. Replacing $\Gamma(-\infty,0)$ with a straight line segment from $(0,0)$ to $(a,0)$ results in an escape strategy ζ.

We can now adapt the two parameters a,β to obtain a strategy ζ that approximates every π_s best. Let $p = (a\cdot e^{\phi\cot(\beta)},\phi)$ be the coordinate where ζ escapes from R; for the certificate path, there are two extreme cases that can occur, see Figure 3.10: (i) Either π_s is a straight line segment, which implies $\alpha_x = 0$ and $x = a\cdot e^{\phi\cot(\beta)}$; (ii) or π_s is a straight line segment attached to a full circle around s, which implies $\alpha_x = 2\pi$ and $x = a\cdot e^{(\phi-2\pi)\cot(\beta)}$. For both cases, we want the spiral strategy to attain

the same approximation factor; hence we set

$$\frac{|\,\zeta(0,\phi)\,|}{a\cdot e^{\phi\cot\beta}(1+0)} = \frac{|\,\zeta(0,\phi)\,|}{a\cdot e^{(\phi-2\pi)\cot\beta}(1+2\pi)} \quad\Leftrightarrow\quad 1 = \frac{e^{2\pi\cot\beta}}{1+2\pi}$$

and obtain $\beta = \operatorname{arccot}(\ln(2\pi+1)/(2\pi)) \approx 1.265$.

Theorem 7. *Let $\mathscr{R}$ be the family of regions with a non-empty kernel. If $s \in \ker(R)$ for $R \in \mathscr{R}$, then the spiral strategy ζ with eccentricity $\beta = \operatorname{arccot}(\ln(2\pi+1)/(2\pi))$ is a c-approximation of the unknown certificate path π_s with $c = 1/\cos\beta \approx 3.318674$.*

Proof. Let π_s be given by $x \in \mathbb{R}_{>0}$ and $\alpha_x \in [0,2\pi]$; with $S(x,\alpha_x)$, we denote the circular sector centred at s of radius x and central angle α_x. Hence, π_s escapes from R after searching $S(x,\alpha_x)$ in the worst case. We first show that ζ escapes from R as soon as ζ covers $S(x,\alpha_x)$: Let t be the first time that ζ covers $S(x,\alpha_x)$, i. e., $t \coloneqq \min_\varphi\{\varphi \mid a\cdot e^{(\varphi-\alpha_x)\cot(\beta)} \geq x\}$; then $p = \zeta(t)$ can be expressed by polar coordinates (r,ϕ) with $r = a\cdot e^{\phi\cot(\beta)}$, see Figure 3.10iii. Consider the coordinates $s_l \coloneqq (x,\phi)$ and $s_r \coloneqq (x,\phi-\alpha_x)$ that frame $S(x,\alpha_x)$. By definition of α_x, the open balls $B_\varepsilon(s_l)$ and $B_\varepsilon(s_r)$ with radius ε contain points of $\partial R\ \forall\varepsilon > 0$; moreover, since $s \in \ker(R)$, ζ can never cross the rays emanating from s through s_l, s_r at a distance larger than x. Consequently, if ζ just passes through $(x,\phi-\alpha_x)$ without intersecting with ∂R, then ζ escapes at p the latest. Finally, we compare the length of ζ up to ϕ to the length of π_s by

$$f(\alpha_x) = \frac{|\,\zeta(0,\phi)\,|}{a\cdot e^{(\phi-\alpha_x)\cot\beta}(1+\alpha_x)} \leq \frac{\frac{a}{\cos\beta}\cdot e^{\phi\cot\beta}}{a\cdot e^{(\phi-\alpha_x)\cot\beta}(1+\alpha_x)} = \frac{e^{\alpha_x\cot\beta}}{\cos\beta(1+\alpha_x)},$$

which attains a maximum value of $1/\cos(\beta)$ for $\alpha_x \in \{0,2\pi\}$. □

Note that such a spiral strategy ζ can never be a constant factor approximation for the family of arbitrary simply-connected regions: For any $c \in \mathbb{R}$, there exists a simply connected region that winds itself around the spiral path such that the approximation factor of ζ is larger than c.

3.4.3 Implications of the Spiral Strategy

The spiral strategy of Theorem 7 approximates π_s with a small approximation factor for regions R where $s \in \ker(R)$. As we show in the following, the strategy also yields a rather small approximation factor for the family of regions with a non-empty kernel.

Corollary 5. *Let $\mathscr{R}$ be the family of regions with a non-empty kernel. The spiral strategy ζ with eccentricity $\beta = \operatorname{arccot}(\ln(2\pi+1)/(2\pi))$ is a c-approximation of the certificate path for $\mathscr{R}$ with $c < 8.1125$.*

Proof. Similar to the proof of Theorem 7, let π_s be given by $x \in \mathbb{R}_{>0}$ and $\alpha_x \in [0,2\pi]$; let $S(x,\alpha_x)$ be the circular sector centred at s of radius x and central angle α_x. Hence, π_s escapes from R after searching $S(x,\alpha_x)$ in the worst case. Moreover, let $p = \zeta(t)$, s_l, s_r be defined as in the proof of Theorem 7; see Figure 3.10iii.

For the proof, we proceed in three steps: In a first step, we identify a point $p' \coloneqq \zeta(\phi+\overline{\varphi})$ where $\overline{\varphi} < 2.8286$ and express p' in polar coordinates. In a second step, we prove via contradiction that ζ escapes from R at time $t \leq \phi+\overline{\varphi}$ in the worst case using that R has a non-empty kernel. Finally, we conclude with proving that the approximation factor holds for every π_s.

First, we define p' via $\overline{\varphi}$ as follows; see Figure 3.11: Let ϕ' be the largest value smaller than ϕ such that the line through p and $\zeta(\phi')$ is tangent to ζ at $\zeta(\phi')$; with g, we denote the line through p and $\zeta(\phi')$. Then, we define $\overline{\varphi}$ as the first intersection of g and ζ larger than ϕ, i. e., $\overline{\varphi} \coloneqq \min_{\varphi>0}\{\varphi \mid g \cap \zeta(\phi+\varphi) \neq \emptyset\}$. W. l. o. g. we may choose the coordinate system such that g is tangent to ζ at $(a,0)$, since the angle $\overline{\varphi}$ remains unchanged; thus, the slope of g equals $\tan(\beta)$ by definition of the eccentricity. The line g can be expressed in slope-intersept form $y = g(x) = \tan(\beta)\cdot x - \tan(\beta)\cdot a$; moreover, expressing ζ in Cartesian coordinates, we obtain the following system of equations which can be solved to obtain the equation on the right hand side:

$$\begin{array}{rlll} g: & y = & \tan\beta\cdot x - \tan\beta\cdot a & \\ \zeta: & x = & a\cdot e^{\varphi\cdot\cot\beta}\cdot\cos\varphi & \Leftrightarrow \qquad 1 = e^{\varphi\cot\beta}\cdot\left(\cos\varphi - \frac{\sin\varphi}{\tan\beta}\right). \\ & y = & a\cdot e^{\varphi\cdot\cot\beta}\cdot\sin\varphi & \end{array}$$

The first two positive zeros of the equation give ϕ and $(\phi+\overline{\varphi})$, which can be computed numerically to obtain $\phi \approx 4.62896, (\phi+\overline{\varphi}) \approx 7.45744$ and $\overline{\varphi} < 2.8286$.

Second, we prove that ζ escapes from R at time $t \leq \phi+\overline{\varphi}$. Assume to the contrary this were not true, then $\zeta(0,\phi+\overline{\varphi}) \cap \partial R = \emptyset$. Since R is simply-connected, $B_\varepsilon(s_l)$ contains a point of $\partial R\ \forall\varepsilon > 0$, and s_l lies closer to s than p, ∂R has to cross the line segment $\overline{pp'}$; thus, the intersection of $\overline{pp'}$ and ∂R is non-empty, i. e., $I \coloneqq \overline{pp'} \cap \partial R \neq \emptyset$. For every $v \in \ker(R)$ we conclude that: $v \notin g$, since boundary-points in I block the visibility between p and p' along g; v cannot lie above g, since boundary-points in I block the visibility to either p or p'; v cannot lie below g, since boundary-points in I block the visibility to points of R above g. Thus, $\ker(R) = \emptyset$ which contradicts the initial assumption.

Finally, we bound the approximation factor for every π_s by comparing the length of $\zeta(0,\phi+\overline{\varphi})$ to the length of π_s by

$$f'(\alpha_x) = \frac{|\,\zeta(0,\phi+\overline{\varphi})\,|}{a\cdot e^{(\phi-\alpha_x)\cot\beta}(1+\alpha_x)} \leq \frac{\frac{a}{\cos\beta}\cdot e^{(\phi+\overline{\varphi})\cot\beta}}{a\cdot e^{(\phi-\alpha_x)\cot\beta}(1+\alpha_x)} = e^{\overline{\varphi}\cdot\cot\beta}\cdot f(\alpha_x),$$

which attains a maximum value of $e^{\overline{\varphi}\cdot\cot\beta}/\cos(\beta) \approx 8.112465$ for $\alpha_x \in \{0, 2\pi\}$. □

Note that the approximation factor can be brought closer to 8 by choosing β slightly larger than $\operatorname{arccot}(\ln(2\pi+1)/(2\pi))$; this, however, weakens the optimal approximation factor for the case where s lies in the kernel.

Finally, the spiral strategy does not only approximate the certificate path with a reasonably small approximation factor. Due to the results of Section 3.3, the results directly transfer to the approximation of *shortest escape paths* for several families of regions.

Corollary 6. *For the family of fat convex shapes, equilateral triangles and infinite strips, the spiral strategy ζ with eccentricity $\beta = \operatorname{arccot}(\ln(2\pi+1)/(2\pi))$ is an* optimal *c-approximation of the* shortest escape path *with $c < 3.3187$.*

Splendidly, the approximation holds even though neither the precise shape of the region[12] nor the concrete size of this shape is known! We expect that the same holds for several other families for which we have not given proofs in Section 3.3.

[12] More precisely, we assume that a given region is known to be of one of the following types: a fat convex shape, an equilateral triangle or an infinite strip. However, the exact type remains unknown.

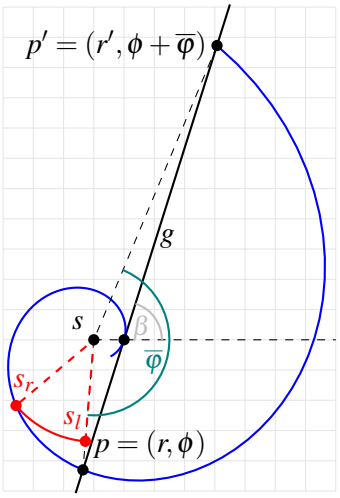


Figure 3.11: The spiral strategy ζ (blue) leaves ∂R at p' in the worst case. At p, ζ has searched the circular sector $S(x, \alpha_x)$ (red) and does not leave R, yet. The point p' is defined via the line through p that is tangent to ζ.

For the family of half-planes, the approximation factor of Isbell's path increases only slightly due to Lemma 7. Again, note that neither the distance from s to the boundary nor the family of regions is known to the escape strategy!

Corollary 7. *For the family of infinite half-planes, the spiral strategy ζ with eccentricity $\beta = \operatorname{arccot}(\ln(2\pi+1)/(2\pi))$ is a c-approximation of Isbell's path with a factor $c < 3.6304$.*

For the sake of completeness, note that Isbell's path can be approximated with a factor of $c < 2.159$ when the family of regions is known to contain only half-planes. This can be achieved with a spiral strategy that directly approximates the distance from s to ∂H with a factor of less than 13.82; e. g., see [8, 37]. Similarly, better approximation factors are conceivable if the family of regions is precisely known to contain only shapes of a specific type, e. g., only fat convex shapes, only equilateral triangles, or only infinite strips.

In general, logarithmic spirals are natural candidates for optimal competitive search strategies, but in almost all cases the optimality remains a conjecture; see [8, 32, 35, 37, 3]. In [73], the optimality of spiral search was shown for the problem of searching for a point in the plane with a radar. Many other conjectures are still open. For example Finch and Zhu [37] considered the problem of searching for a line in the plane; Finch [35] conjectures that the family of logarithmic spirals contains an optimal strategy. In the following section, we develop a lower bound which establishes optimality of the approximation factors given in Theorem 7 and Corollary 6.

3.5 A Lower Bound for the Approximation Factor

Throughout this section, $\mathscr{R}$ denotes the family of regions with a non-empty kernel. We prove the following lower bound for the approximation of the certificate path for $\mathscr{R}$.

Theorem 8. *For the family of regions $\mathscr{R}$ with a non-empty kernel, there is no escape strategy that approximates the certificate path for $\mathscr{R}$ with a factor of less than* 3.3126.

The idea of the proof is as follows. Given a c-approximation of the certificate path, we first prove that c can be bounded from below by discrete functionals on positive, real-valued sequences. To establish the lower bound of the approximation factor by discrete functionals, we proceed in three steps: With Lemma 9, we show how an arbitrary continuous escape strategy can be discretised to

obtain a sequence of coordinates on rays. The sequence of coordinates is used to construct tight upper bounds for the length of the certificate path in Lemma 10. Finally, Lemma 11 allows us to rearrange the positive, real-valued sequences obtained from the coordinates and thus relate the strategy's cost to the cost of π_s. Then, using a theorem of Gal, we optimize the sum of two extreme functionals to obtain a lower bound for the maximum of both.

Lemma 9 (discretisation). *Let ζ be a c-approximation of the certificate path for $\mathscr{R}$ and $n \geq 3$ be an odd number of equidistant rays $r_1, r_2, \ldots, r_n$ emanating from $s = \zeta(0)$.*

Then, there exists an infinite sequence $Z = \{z_i\}_{i\geq 0}$ of tuples $z_i := (x_i, N_i)$ such that:

1. *$x_i > 0$ denotes the search depth along the ray with index $N_i \in \{1, \ldots, n\}$ in the i-th step;*
2. *any two consecutive coordinates z_i, z_{i+1} lie on neighbouring rays; and*
3. *for every $k \geq 1$ with $z_k = \zeta(t_k)$ for some $t_k > 0$, it is $|z_0| + \sum_{i=0}^{k-1} d(z_i, z_{i+1}) \leq \zeta(0, t_k)$.*

Proof. Let the n rays emanating from $s = \zeta(0)$ be denoted with $r_1, r_2, \ldots, r_n$ in clockwise order; for this proof, we assume that s is not part of the rays, i. e. $s \notin r_j \forall j$.

Since ζ achieves a finite, constant approximation factor for any certificate path, it must *visit* all of the rays infinitely many often[13], i. e., $\forall j \forall t > 0 \exists t' > t : \zeta(t') \in \{r_j\}$. Moreover, for each ray r_j visited at time t, there is a different ray that ζ visits afterwards, i. e., if $\zeta(t) \in \{r_j\}$, then $\exists t' > t \exists k \neq j : \zeta(t') \in \{r_k\}$. Now we obtain the entries of Z by the following procedure.

Let r_j be the ray that ζ visits first. We define x_0 as the largest distance from s that ζ visits on r_j before another ray is visited, i. e.

$$x_0 := \max\{\, t > 0 \;\big|\; (\zeta(t) \in \{r_j\}) \wedge (\forall k \neq j : \{\zeta(0,t)\} \cap \{r_k\} = \emptyset)\,\};$$

since ζ visits another ray afterwards, this maximum value must exist. Altogether this defines $z_0 = (x_0, N_0)$ where $N_0 = j$ is the index of the ray visited first.

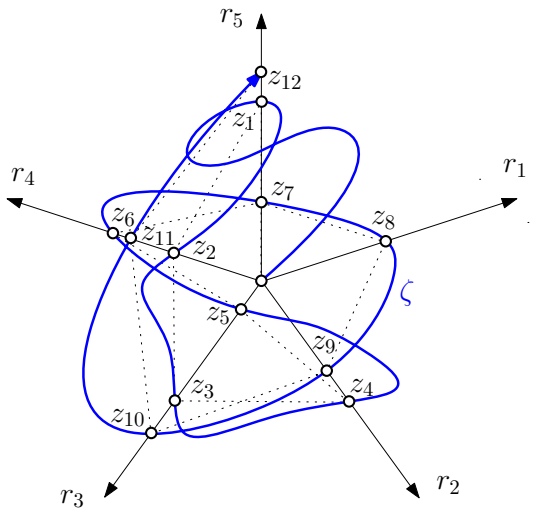
Let t_{i-1} be the time of the visit registered in the $(i-1)$-th step of Z; i. e., $\zeta(t_{i-1}) = z_{i-1}$. We define the i-th entry of Z for $i > 0$ inductively by the following procedure: Let r_j be the ray visited next, which implies that $j = N_i \neq N_{i-1}$; then, x_i is the largest distance visited along r_j by ζ after leaving the previous ray and before visiting another ray, i. e.

$$x_i := \max\{\, t > t_{i-1} \;\big|\; (\zeta(t) \in \{r_j\}) \wedge (\forall k \notin \{j, N_{i-1}\} : \{\zeta(t_{i-1}, t)\} \cap \{r_k\} = \emptyset)\,\}.$$

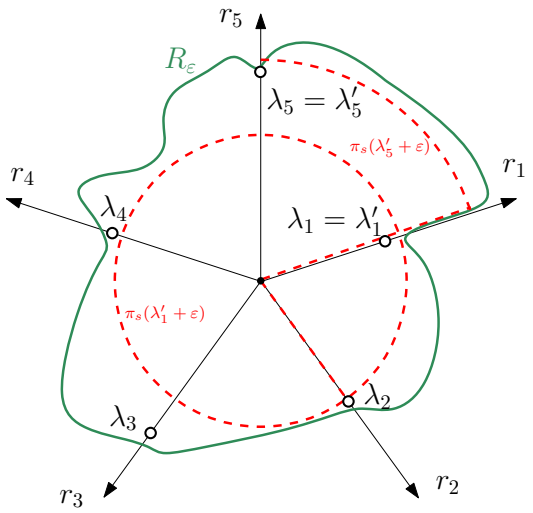
Again, this maximum value has to exist since ζ visits another ray afterwards. Hence, we obtain an infinite sequence of coordinates Z from ζ, see Figure 3.12i.

Note that this construction does not imply that z_i and z_{i+1} lie on neighbouring rays: ζ can pass through s and visit an arbitrary ray at any time. To ensure that this property holds, we employ the following trick: Between any two entries z_i, z_{i-1} of Z, the direct edge from z_{i-1} to z_i intersects at most $(\lfloor n/2 \rfloor - 1)$ rays and never s, since n is odd and the rays are equidistant; hence, we can add the coordinates of these intersection points to Z.

Finally, for every $\zeta(t_k) = z_k$, the sum of straight-line distances between every pair of consecutive points of $\{z_i\}_{i=0}^{k}$ is never longer than the length of $|\zeta(0, t_k)|$. This can be easily proven via induction over k applying the triangle inequality in the induction step. □



(i) Discretisation of a c-approximation ζ (blue) along 5 rays. Only the first 12 entries of Z are given.



(ii) In region R_ε, every $\pi_s(\lambda'_j + \varepsilon)$ (red), is an escape path and an upper bound for π_s; e. g., for $j \in \{1,5\}$.

Figure 3.12: (i) Every c-approximation of the certificate path for $\mathscr{R}$ can be discretised to obtain an infinite sequence of coordinates. (ii) At any time, the maximal search depths can be used to construct concrete regions for which bounds to the certificate paths are known.

In the following, we assume that ζ escapes from some region R at time t. By the discretisation of the previous lemma, we obtain a finite, discrete sequence $Z_k = \{z_i\}_{i\geq 0}^k$ such that $z_k = \zeta(t)$ is the point where ζ escapes from R. With the following lemma, we construct special regions around $\zeta(0,t)$ such that we are able to give a tight upper bound for the length of the certificate path in each of these regions by search depths of coordinates of Z_k. Therefore, let λ_j denote the maximum search depth that Z_k attains on the j-th ray, i. e., $\lambda_j := \max_{0\leq i\leq k}\{x_i \mid N_i = j\}$; see Figure 3.12ii. Finally, let $\Lambda := \{\lambda_1, \ldots, \lambda_n\}$ be the set of all maximum search depths and $\Lambda' = (\lambda'_1, \ldots, \lambda'_n)$ be a permutation of Λ such that the search depths are sorted in descending order; i. e., $\lambda'_1 \geq \lambda'_2 \geq \cdots \geq \lambda'_n$.

Lemma 10 (certificate path upper bound). *Given an escape strategy ζ that escapes at time t from a region $R \in \mathscr{R}$. Let Z_k be a discretization of $\zeta(0,t)$ on n equidistant rays as described above; moreover, let $\Lambda' = (\lambda'_1, \ldots, \lambda'_n)$ be the sequence of maximal search depths in descending order.*

For every $\varepsilon > 0$, there exists a region $R_\varepsilon \in \mathscr{R}$ such that $|\pi_s| \leq (\lambda'_j + \varepsilon) \cdot (1 + j \cdot 2\pi/n)$ for every $1 \leq j \leq n$; moreover, ζ escapes from R_ε not earlier than time t.

Proof. Let s denote the common origin of the n rays. Due to equidistance, the angle between neighbouring rays is equal to $2\pi/n$. We construct R_ε as follows: First, set the distance from s to ∂R_ε along ray r_i to λ_i, where $i = N_k$; this ensures that ζ still escapes at time t. Then, for every ray r_i with index $i \neq N_k$, set the distance from s to the boundary of R_ε along r_i to $\lambda_i + \varepsilon$; connect the points of ∂R_ε on the rays such that $s \in \ker(R_\varepsilon)$ and ∂R_ε neither intersects with $\zeta(0,t)$ nor with the polygonal chain defined by Z_k, see Figure 3.12i.

[13] Assume to the contrary this were not the case. Then, one of the rays is never visited after a certain time and beyond a certain radius. Since the boundary could hide along this ray, the approximation factor can never be constant which results in a contradiction.

Since, by construction, there are no additional intersections of ζ with R_ε, ζ escapes from R_ε at time t. Moreover, $\pi_s(\lambda'_j+\varepsilon)$ consists of a straight line segment of length $x := (\lambda'_j+\varepsilon)$ attached to an circular arc segment of length $x\cdot\alpha_x$ around s. By a similar reasoning as in Section 3.2.2, $\pi_s(x)$ visits at most $(j+1)$ consecutive rays at depth $(\lambda'_j+\varepsilon)$ until it reaches ∂R_ε along one of which; thus, $\alpha_x \leq j\cdot 2\pi/n$. Hence, in R_ε, $\pi_s(x)$ is an escape path and $|\,\pi_s\,| \leq x\cdot(1+\alpha_x)$. □

Lemma 11 (reordering sequences). *Let $\theta\in\mathbb{R}$ and $d:\mathbb{R}^2\to\mathbb{R}$ with $d(x,y)\mapsto\sqrt{x^2-2xy\cos\theta+y^2}$. Given a finite, positive sequence $X_k=\{x_i\}_{i\geq 0}^k$ and let $\overrightarrow{X}_k$ be a sequence obtained from X_k by rearranging the entries in monotonically increasing order. Then $x_0+\sum_{i=0}^{k-1} d(x_i,x_{i+1}) \geq \overrightarrow{x}_0 + \sum_{i=0}^{k-1} d(\overrightarrow{x}_i,\overrightarrow{x}_{i+1}) \geq \sum_{i=0}^{k-1} d(\overrightarrow{x}_i,\overrightarrow{x}_{i+1})$ holds.*

Proof (via induction). We prove the first inequality via induction over k, the second inequality holds since all x_i are positive. For the induction start $k=1$, we have $\overrightarrow{x}_0=\min(x_0,x_1)$ and the inequality holds due to the symmetry of d.

For the induction step $k\to k+1$, we define x_m as the maximum of X_k with the largest index, i. e., m is the largest index such that $x_i\leq x_m$ holds for all $0\leq i\leq k$. W. l. o. g. we may assume that $0<m<k$: If $m=0$ we first reverse the sequence to obtain $\overline{X}_k$, where $\bar{x}_i := x_{k-i}$ and $\bar{x}_0 = x_k < x_m = x_0$; due to the symmetry of d, it is $x_m+\sum_{i=0}^k d(x_i,x_{i+1}) > \bar{x}_0+\sum_{i=0}^k d(\bar{x}_{k-i},\bar{x}_{k-i+1}) = \bar{x}_0+\sum_{j=0}^k d(\bar{x}_j,\bar{x}_{j+1})$. If $m=k$, the inequality can be directly proven by applying the induction hypothesis to $x_0+\sum_{i=0}^{k-1} d(x_i,x_{i+1})$ since $x_m=x_k=\overrightarrow{x}_k$. It remains to consider the case where $0<m<k$; we define $y_i := x_i$ for $0\leq i\leq m-1$, $y_i := x_{i+1}$ for $m\leq i\leq k$, $y_{k+1} := x_m = \overrightarrow{y}_{k+1}$, $a := \min(x_{m-1},x_{m+1})$, $b := \max(x_{m-1},x_{m+1})$ to obtain

$$
\begin{aligned}
x_0+\sum_{i=0}^{k} d(x_i,x_{i+1}) &= x_0+\sum_{i=0}^{m-2} d(x_i,x_{i+1}) + d(x_{m-1},x_m) + d(x_m,x_{m+1}) \\
&\quad + \sum_{i=m+1}^{k} d(x_i,x_{i+1}) + d(x_{m-1},x_{m+1}) - d(x_{m-1},x_{m+1}) \\
&= y_0+\sum_{i=0}^{m-2} d(y_i,y_{i+1}) + d(a,x_m) + d(b,x_m) \\
&\quad + \sum_{i=m+1}^{k} d(y_{i-1},y_i) + d(y_{m-1},y_m) - d(a,b) \\
&= y_0+\sum_{i=0}^{k-1} d(y_i,y_{i+1}) + d(a,x_m) + d(b,x_m) - d(a,b) \\
&\overset{\text{IH}}{\geq} \overrightarrow{y}_0+\sum_{i=0}^{k-1} d(\overrightarrow{y}_i,\overrightarrow{y}_{i+1}) + d(a,x_m) + d(b,x_m) - d(a,b) \\
&= \overrightarrow{y}_0+\sum_{i=0}^{k} d(\overrightarrow{y}_i,\overrightarrow{y}_{i+1}) - d(\overrightarrow{y}_k,x_m) + d(a,x_m) + d(b,x_m) - d(a,b) \\
&\geq \overrightarrow{y}_0+\sum_{i=0}^{k} d(\overrightarrow{y}_i,\overrightarrow{y}_{i+1}) = \overrightarrow{x}_0+\sum_{i=0}^{k} d(\overrightarrow{x}_i,\overrightarrow{x}_{i+1}).
\end{aligned}
$$

Since x_m is a maximum of X_k and $\overrightarrow{y}_k$ is a maximum of $X_k\setminus\{x_m\}$, we have $a\leq b\leq\overrightarrow{y}_k\leq x_m$; thus, the last inequality holds due to Lemma 24 part 2 which we give in the Appendix B.2. □

Finally, we prove the lower bound.

Proof of Theorem 8. Let ζ be a c-approximation of the certificate path for $\mathscr{R}$. By definition, for arbitrary $R \in \mathscr{R}$ with $s \in R$, $\zeta(t)$ escapes at some time t and $|\,\zeta(0,t)\,| \leq c \cdot |\,\pi_s\,|$ holds. By Lemma 9, for any odd $n \geq 3$, ζ can be discretised to obtain a positive sequence $Z_k = \{z_i\}_{i=0}^{k}$, where $\zeta(t) = z_k$ and consecutive entries correspond to visits of neighbouring rays. As before, d denotes the Euclidean metric.

Since ζ escapes at z_k, we can bound c from below with the following chain of inequalities for any $1 \leq j \leq n$:

$$c \geq \frac{|\,\zeta(0,t)\,|}{|\,\pi_s\,|} \geq \frac{x_0 + \sum_{i=0}^{k-1} d\,(z_i, z_{i+1})}{|\,\pi_s\,|} \tag{3.1a}$$

$$= \frac{x_0 + \sum_{i=0}^{k-1} \sqrt{x_i^2 - 2\cos\left(\frac{2\pi}{n}\right) x_i x_{i+1} + x_{i+1}^2}}{|\,\pi_s\,|} \tag{3.1b}$$

$$\geq \frac{x_0 + \sum_{i=0}^{k-1} \sqrt{x_i^2 - 2\cos\left(\frac{2\pi}{n}\right) x_i x_{i+1} + x_{i+1}^2}}{(\lambda'_j + \varepsilon)(1 + j \cdot \frac{2\pi}{n})} \tag{3.1c}$$

$$\geq \frac{\overrightarrow{x}_0 + \sum_{i=0}^{k-1} \sqrt{\overrightarrow{x}_i^2 - 2\cos\left(\frac{2\pi}{n}\right) \overrightarrow{x}_i \overrightarrow{x}_{i+1} + \overrightarrow{x}_{i+1}^2}}{(\overrightarrow{x}_{k+1-j} + \varepsilon) \cdot (1 + j \cdot \frac{2\pi}{n})} \tag{3.1d}$$

$$\geq \frac{\sum_{i=0}^{k-1} \sqrt{\overrightarrow{x}_i^2 - 2\cos\left(\frac{2\pi}{n}\right) \overrightarrow{x}_i \overrightarrow{x}_{i+1} + \overrightarrow{x}_{i+1}^2}}{\overrightarrow{x}_{k+1-j} \cdot (1 + j \cdot \frac{2\pi}{n})}. \tag{3.1e}$$

The second inequality in 3.1a holds due to Lemma 9 part 3. Equality 3.1b holds by Lemma 9 part 2 since consecutive entries in Z lie on neighbouring, equidistant rays. Since $\zeta(t)$ is a c-approximation for every region in $\mathscr{R}$, inequality 3.1c also holds for the special region $R_\varepsilon \in \mathscr{R}$ of Lemma 10. Inequality 3.1d holds due to Lemma 11 and since the j-th largest maximum search depth λ'_j is exactly $\overrightarrow{x}_{k+1-j}$ in the reordered sequence. Finally, in the last inequality we may leave out $\overrightarrow{x}_0$ and ε since ε can be chosen as small as necessary[14].

Since this bound holds for every k, this leads us to the definition of a sequence of functionals

$$F_l^j(X) := \frac{\sum_{i=-l}^{k-1} \sqrt{x_i^2 - 2\cos\left(\frac{2\pi}{n}\right) x_i x_{i+1} + x_{i+1}^2}}{x_{k+1-j}(1 + j \cdot \frac{2\pi}{n})} \quad \text{for } l \geq k \geq n \quad \text{and}$$

$$F^j(X) := \lim_{l \to \infty} F_l^j(X) = \frac{\sum_{i=-\infty}^{k-1} \sqrt{x_i^2 - 2\cos\left(\frac{2\pi}{n}\right) x_i x_{i+1} + x_{i+1}^2}}{x_{k+1-j}(1 + j \cdot \frac{2\pi}{n})},$$

where X denotes a doubly infinite, positive sequence.[15] Altogether, for every odd $n \geq 3$, every $1 \leq j \leq n$ and every $k \geq n$, c is bounded from below by the discrete functional $F^j(X)$ where X is a

[14] Namely, $\varepsilon \leq \overrightarrow{x}_0 \overrightarrow{x}_{k+1-j} / \sum_{i=0}^{k-1} \sqrt{\overrightarrow{x}_i^2 - 2\cos\left(\frac{2\pi}{n}\right) \overrightarrow{x}_i \overrightarrow{x}_{i+1} + \overrightarrow{x}_{i+1}^2}$.

[15] The single infinite, positive sequence can be extended to the left by any converging, positive sequence.

positive sequence. Moreover, each functional satisfies the unimodality condition [3, (7.7), p. 109], for which we provide a proof in Section B.1.

Assume that ζ achieves an optimal approximation factor c. For every k, using a theorem by Gal [3, Theorem 7.1, p. 108], X_k can be made arbitrarily close to a geometric sequence by linear combinations of subsequences multiplied with non-negative coefficients; hence, we may as well assume that X_k is a geometric sequence, i. e., $x_i = a^i$ for some $0 \leq a \leq \infty$. Moreover, since each functional F_l^j is unimodal, a lemma by Gal [3, Lemma 7.5, p. 110] shows that the value of the functional does not increase when applied to this linear combination; this implies that it still lower bounds c.

Since X_k is obtained from a strategy with an optimal approximation factor, it minimizes the maximum value of all $F^j(X)$; conversely, when minimizing the sum of the two functionals $F^1(X)$ and $F^n(X)$, we may conclude that one of them bounds c from below with at least one half of the sums value. Hence, we consider $F^1(A) + F^n(A)$ for which we resolve the geometric series part and simplify the expression to obtain

$$g_n(a) \coloneqq \frac{1}{a-1}\left(\frac{\sqrt{1-2\cos\left(\frac{2\pi}{n}\right)a+a^2}}{(1+\frac{2\pi}{n})}\right) + \frac{a^{n-1}}{a-1}\left(\frac{\sqrt{1-2\cos\left(\frac{2\pi}{n}\right)a+a^2}}{(1+2\pi)}\right).$$

For any odd number of rays n, there is a value of $a > 1$ that minimizes g_n; moreover, any such minimal value of g_n gives a lower bound on the sum of two ratios in the original problem. With growing n, $\inf_{a>1} g_n(a)$ converges to ≈ 6.626252 from below; e. g. for $n = 210000001$, we obtain $a \approx 1.00000000907908$ and $g_n(a) \approx 6.626252$, which gives the lower bound of $6.626252/2 = 3.313126$ for c. □

3.5.1 Further Remarks and Discussion of the Lower Bound Technique

In the proof of Theorem 8, Gal's minimax theorem was not applied directly, which has the following reason. As shown in the proof, for every $1 \leq j \leq n$, the approximation factor is bounded from below by a sequence of functionals $F_l^j(X)$ with the functional $F^j(X)$ in the limit. These functionals satisfy all necessary preconditions[16] of Gal's minimax theorem which we verify in Section B.1. Hence, we may apply Gal's theorem [3, Theorem 7.9] to conclude that the functional attains a minimal value when applied to a geometric sequence A_a. More precisely, we obtain that

$$\begin{aligned}
\limsup_{i\to\infty} F^j(X^{+i}) &\geq \inf_{0\leq a\leq\infty} F^j(A_a) = \inf_{a>1} \frac{\sum_{i=-\infty}^{k-1}\sqrt{a^{2i}-2\cos\left(\frac{2\pi}{n}\right)a^{2i+1}+a^{2i+2}}}{a^{k+1-j}(1+j\cdot\frac{2\pi}{n})} \\
&= \inf_{a>1} \frac{\sum_{i=-\infty}^{k-1} a^i\sqrt{1-2\cos\left(\frac{2\pi}{n}\right)a+a^2}}{a^{k+1-j}(1+j\cdot\frac{2\pi}{n})} = \inf_{a>1} \frac{\sqrt{1-2\cos\left(\frac{2\pi}{n}\right)a+a^2}}{(1+j\cdot\frac{2\pi}{n})}\cdot\frac{\sum_{i=-k+1}^{\infty} a^{-i}}{a^{k+1-j}} \\
&= \inf_{a>1} \frac{\sqrt{1-2\cos\left(\frac{2\pi}{n}\right)a+a^2}}{(1+j\cdot\frac{2\pi}{n})}\cdot\frac{\sum_{i=0}^{\infty} a^{-i}+\sum_{i=1}^{k-1} a^i}{a^{k+1-j}} = \inf_{a>1} \frac{\sqrt{1-2\cos\left(\frac{2\pi}{n}\right)a+a^2}}{(1+j\cdot\frac{2\pi}{n})}\cdot\frac{\frac{a}{a-1}+\left(\frac{1-a^k}{1-a}-1\right)}{a^{k+1-j}} \\
&= \inf_{a>1} \frac{\sqrt{1-2\cos\left(\frac{2\pi}{n}\right)a+a^2}}{(1+j\cdot\frac{2\pi}{n})}\cdot\frac{a^k}{a^{k+1-j}(a-1)} = \inf_{a>1} \frac{a^{j-1}}{(1+j\cdot\frac{2\pi}{n})}\cdot\frac{\sqrt{1-2\cos\left(\frac{2\pi}{n}\right)a+a^2}}{a-1},
\end{aligned}$$

where X^{+i} denotes the sequence obtained from X by shifting indices i to the right. For growing values of n, the infimum and the corresponding value of $a > 1$ tends to the values as given in Table 3.1. Since

[16] These are given in [3, (7.5)-(7.9), p. 110] and [3, (7.20), p. 113].

$j=$	1	2	3	...	$n-1$	n
$\overline{a}\approx$	∞	1.0073443	1.0058163	...	1.0000977	1.0000976
$F^j(A_{\overline{a}})\approx$	1	1.0097814	1.0156709	...	2.3736137	2.3736407

Table 3.1: For fixed j and $n = 10001$, the infimum is attained for the value $\overline{a} := \arg\min_{0\leq a\leq\infty} F^j(A_{\overline{a}})$ and given in the corresponding column. All values are rounded to the last decimal; for larger values of n, only positions far after the decimal point are subject to minor changes. Clearly, the values exemplify the transition from DFS to BFS: For $j = 1$, an infinite base $\overline{a}$ for the exponential strategy $A_{\overline{a}}$ defines a pure DFS with an optimal approximation factor of 1; for larger values of n, the value of $\overline{a}$ quickly approaches 1 causing a growth of the approximation factor.

every $F^j(A_a)$ gives a lower bounds on c that is far from the upper bound of Theorem 7, the sum of two extreme functionals is used to prove the lower bound.

Comparing the bounds on the approximation factor given by Theorem 7 and 8, one observes a small gap of roughly 0.0056 which can be explained as follows. In the proof the lower bound, the sum of the two extreme functionals F^1 and F^n is considered and minimized to obtain a lower bound for the maximum of one of them. Consider the values for n and a used in the proof of the theorem; evaluating the two functionals F^1 and F^n separately for these values reveals that $F^1(A_a) \geq 3.4438$ and $F^n(A_a) \geq 3.1823$. Hence, minimizing their sum does not imply that both functionals attain the same value; this weakens the lower bound and explains why a small gap remains.

The gap can be closed with the following approach. Instead of the sum $F^1(X) + F^n(X)$ of the extreme functionals, we consider their weighted sum $p/q \cdot F^1(X) + F^n(X)$ and minimize the corresponding function $\tilde{g}_n(a)$. For example, $\tilde{g}_n(a)$ attains a minimum of $\tilde{g}_n(a) \approx 7.236546$ for $a \approx 1.000000198$ and the choice $p = 85, q = 72$. This translates[17] into a lower bound of greater than ≈ 3.31867 and closes the gap. In contrast to the values of $F^1(A_a)$ and $F^n(A_a)$ as before, the new value of a almost balances the difference in the values of the functionals. The gap can be further closed by adjusting the multiplicative factor to perfectly balance the value of these functionals.

For other geometric search problems, the lower bound technique presented above can be applied to prove optimality of a given strategy. To do so, one can challenge the given strategy w. r. t. the approximation of the certificate path π_s: An upper bound for the approximation factor can often be derived from the given strategy, since π_s (or rather its length) is easy to bound from above; to prove a lower bound, the technique used in the proof of Theorem 8 can be used as a framework. Note that, for the problem at hand, the framework can not used as a black box. Quite the contrary: The exact combination of the Lemmata 9-11 will not always suffice to establish the lower bound by discrete functionals; more precisely, it might be necessary to replace some of which by more suitable ones taking the specific search problem into account. However, the rough steps indicated by the lemmata can be used as a guideline to establish the desired connection: (i) Derive a *discretization* of the strategy as a real-valued sequence, (ii) prove *tight upper bounds* for π_s and (iii) rearrange the sequence to establish a connection between the discrete strategy and the certificate path. Finally, optimality of the given strategy follows based on the hypothesis that the certificate path is the best one can do with the given amount of information.

In the following, we exemplify the use of the technique with a proof of optimality for a spiral strategy that searches for a target point in the plane with a radar. Similar to an escape strategy, a

[17] We use the following observation, which can be easily proven via contradiction: *If* $(p/q \cdot X + Y) = c$, *then* $\max\{X,Y\} \geq (cq)/(p+q)$.

Figure 3.13: The problem of searching for a point in the plane with a radar. (i) A search trajectory finds p at time t using a radar. The problem is equivalent to the search for a boundary point if ∂R is a ray; the length of the certificate path is upper-bounded by $\pi_s(x)$ with $\alpha_x = 2\pi$ and $x = |sp|$, see (ii).

search strategy is defined as a trajectory $\zeta : \mathbb{R}_{\geq 0} \to \mathbb{R}^2$ originating from the searcher's starting position s in the Euclidean plane. The target point p is considered as found at time t if the straight line segment from s to $\zeta(t)$ contains p; see Figure 3.13i. Gal [41, 3] analysed the problem in a competitive framework where $|\zeta(0,t)|$ is compared to the straight line distance d from s to p; he showed that with a certain logarithmic spiral search strategy one achieves a competitive ratio of approximately 17.289. In 2010, Langetepe [73] proved that this is optimal with a technique similar to the lower bound technique used above. Using the notation and the results established in this chapter, we can now give a short and elegant prove for the optimality.

Corollary 8. *For the problem of searching a point in the plane, the logarithmic spiral with eccentricity* $\operatorname{arccot}(b)$ *defines a search strategy with an optimal competitive ratio of* $\min_{x>0} e^{2\pi x}\sqrt{1+1/x^2}$*, where* b *is the value that minimizes the competitive ratio.*

Proof. The problem can be considered as an escape path problem where R is the Euclidean plane and ∂R is defined as the ray emanating from p into the direction of $\overrightarrow{sp}$; see Figure 3.13ii.

An upper bound for the approximation factor can be easily obtained from Gal's spiral searching strategy, cf. [41, pp. 172 ff.] or [3, pp. 157 ff.]. Let d denote the distance from s to p. The path that consists of a straight line segment of length d attached to a full circle of radius d around s is a tight upper bound for π_s. Since the spiral achieves a competitive ratio of less than 17.28935 compared to the straight line distance d, comparing the strategy to the length of π_s one obtains an upper bound of $17.289/(1+2\pi) \approx 2.3739$.

For the lower bound, we apply the technique as presented in this section. Without the need to make modifications, we may use the Lemma 9-11 to bound the approximation factor c from below by a sequence of functionals as done in the proof of Theorem 8; thus, for every odd number of rays $n \geq 3$, we obtain that $c \geq F^n(X)$ where X is a positive sequence. Applying Gal's minimax theorem as stated above, this gives a lower bound on c of greater than 2.3736; see Table 3.1. □

Note that the lower bound given in Theorem 8 does not hold if the shape of regions is restricted to a special type; e. g. for the case where R is the half-plane. If the type of region is restricted, it is not always possible to construct (tight) upper bounds for the length of the certificate path in Lemma 10. In the absence of an upper bound on $|\pi_s|$, it is impossible to establish the lower bound of the approximation factor c by functionals. If the bound is not tight, the technique can only give a weak lower bound on c.

3.6 Conclusion and Outlook

In this chapter, we considered the problem of searching for the boundary of a simply connected region in the plane. Motivated by a similar approach of Kirkpatrick, we introduced a new measure for the intrinsic complexity of instances this geometric search problem: the certificate path π_s. It provided the basis for a competitive analysis comparing search strategies to π_s. We provided a strategy of expanding concentric circles which approximates π_s with a factor of roughly 27 for the family of simply connected regions. For regions with a non-empty kernel, we provided a spiral strategy that approximates π_s with a factor of 8.1125 and with an optimal factor of 3.3187 for the case where $s \in \ker(R)$. Moreover, we provided a matching lower bound on the approximation factor and discussed what the results imply for Bellman's escape path problem.

The problem of searching for the boundary of a region is closely related to Bellman's escape path problem. Unfortunately, exact solutions to the escape path problem are only known for some specific types of regions. We showed that the shortest escape path can be approximated with a small, constant factor for the family of regions which consists of all fat convex shapes, all equilateral triangles and all infinite strips. For these special types of regions, we had to prove that the certificate path actually approximates the shortest escape path; thus, other (convex) geometric shapes can be easily included into this family by proving that the certificate path approximates a shortest escape path. In general, we raised the following two claims: For every simply connected region R, (i) π_s approximates the shortest escape path with a factor $c \leq 1$; and (ii) π_s is almost optimal when given the distance from s to ∂R into every direction.

This chapter raises several open problems. A crucial open problem remains to prove or disprove the two claims (i) and (ii). At least for claim (ii), a proof seems to be within the realms of possibility. Moreover, it remains open to give a strategy which achieves a smallest approximation factor for regions with a non-empty kernel; note that our logarithmic spiral strategy can be improved giving up the constraint that the approximation factor is optimal when $s \in \ker(R)$. The same question remains open for the family of simply connected regions. In both cases, it remains open to provide a (better) lower bound. Finally, it would be interesting to see the technique applied to other geometric search problems in order to prove optimality of certain strategies. For the problem of searching a line in the plane, our lower bound technique seems like a promising approach.

Chapter 4

Shortest-Path-Preserving Rounding

This chapter examines the following *discrete* problem where *multiple* targets have to be met: Given an undirected, edge-weighted graph $G = (V, E, \omega)$ with weight function $\omega : E \to \mathbb{R}_{\geq 0}$; (how efficient) is it possible to find an integer-valued weight function $\widetilde{\omega}$ such that the absolute change in weight of each shortest path is less than ε? What happens if we further add the restriction that shortest paths w. r. t. ω should remain shortest w. r. t. $\widetilde{\omega}$ and vice versa? For which graph classes can we compute such a rounding of the weights efficiently? In this chapter we prove, by reduction from 3-SAT, that the problem is NP-hard in general. For the case where the graph is a tree with n vertices, we present an algorithm which computes a solution in $O(n^2)$ time.

4.1 Formal Problem Statement

Various applications of graphs, in particular applications related to finding shortest[1] paths, naturally get inputs with real weights on the edges; however, for theoretical or practical reasons, graphs with small integer weights are often preferred or even required.

For a practical example, consider a transportation network modelled as an undirected graph with a weight function on the edges that represents the time (or cost) it takes to travel each edge. If the weights are small integers, one could draw a zone map of the network such that the number of zone boundaries crossed by each shortest path corresponds to the weight of the path [48]. Given a graph with real edge weights, we would therefore like to normalize the weights such that weight 1 corresponds to the intended zone diameter of the map, and then round the normalized weights to integers such that shortest paths are maintained. The corresponding zone map would then provide a fairly accurate representation of travel costs, and would be easier to read and use than a map in which the true travel costs are written in full detail next to each edge.

There are also other applications that could take advantage of graphs with integer weights. These include shortest path problems for which there are algorithms that are more efficient with small integer weights than with arbitrary, real weights [47, 98, 99, 102]. Funke and Storandt [40] cite space efficiency, the speed of arithmetic operations, and stability as advantages of low-precision edge weights.

However, rounding edge weights to integer values has also certain drawbacks. As argued and demonstrated by Funke and Storandt [40, 96], naively rounding weights to the nearest integer values can lead to rounding errors accumulating; large rounding errors, in turn, can change the structure of

[1] That is to say a path of minimum weight.

shortest paths. When rounding weights naively, some paths may see their lengths doubled whereas other, arbitrarily long paths may see their lengths reduced to zero [40]. Funke and Storandt argue that also randomized rounding is likely to cause unacceptable errors in any graph that is large enough [40]. This brings us to the following problem statement.

Definition 10 (ε-roundings). *Let $G = (V, E, \omega)$ be an finite, undirected, edge-weighted graph with weight function $\omega : E \to \mathbb{R}_{\geq 0}$. We call $\widetilde{\omega} : E \to \mathbb{N}_0$ a* path-oblivious ε-rounding *on G if the following condition holds:*

1. *For any shortest path π in G, we have $|\,\widetilde{\omega}(\pi) - \omega(\pi)\,| < \varepsilon$, i. e., between ω and $\widetilde{\omega}$, the weight of any shortest path in G changes by strictly less than ε.*

We call $\widetilde{\omega}$ a weak ε-rounding *if, in addition, the following condition holds:*

2. *Any shortest path in $G = (V, E, \omega)$ is also a shortest path in $\widetilde{G} = (V, E, \widetilde{\omega})$.*

Moreover, we call $\widetilde{\omega}$ a strong ε-rounding *if it is a weak ε-rounding and, additionally, the following condition is satisfied:*

3. *Any shortest path in $\widetilde{G}$ is also a shortest path in G.*

The definition raises the question whether these types of ε-roundings always exists for a sufficiently large choice of the error threshold ε. For the case of path-oblivious and weak ε-roundings, it is not difficult to see that for any graph G there exists an ε such that G admits a trivial weak ε-rounding: We could simply choose ε to be larger than the diameter of G and round all weights down to zero. Thus, the loss of weight along each shortest path is less than ε; moreover, shortest paths remain shortest, since all paths are of length zero, then.

For many other types of problems, a huge choice of ε, resulting in huge differences in weight, would make the concept of rounding moot. We would rather have an ε-rounding for a small value of ε such as $\varepsilon = 1$, but in that case, an ε-rounding does not always exist. For example, consider a star that consists of three edges of weight $1/2$ does not admit a 1-rounding: at least two of the three edges would have to be rounded in the same way. If we round two edges down, there would be a shortest path with rounding error -1; if we round two edges up, there would be a shortest path with rounding error 1. Given an undirected graph $G = (V, E, \omega)$ with non-negative real weight function $\omega : E \to \mathbb{R}_{\geq 0}$ and a error tolerance ε, our problem is therefore to decide whether G admits a path-oblivious, weak, or strong ε-rounding.

Organisation of this chapter. In the remainder of this section, we examine whether strong ε-roundings always exist for large choices of ε and discuss research related to our problem. Section 4.2 settles the complexity status of this problem by proving that all three versions of the decision problem are NP-hard for general graphs. In Section 4.3, we present an algorithm which solves the problem in $O(n^2)$ time for trees with n vertices. In fact, trees always admit a 2-rounding and we can compute, in $O(n^2 \log n)$ time, the smallest ε such that the tree admits an ε'-rounding for any $\varepsilon' > \varepsilon$. The algorithm is constructive, that is, it can easily be adapted to produce the corresponding weights $\widetilde{\omega}$. We conclude with Section 4.4.

4.1.1 On the Existence of Strong ε-Roundings

For the case of strong ε-roundings, we can easily prove their existence if the weight function is restricted to non-negative rational weights:

Observation 4 (Existence of strong ε-roundings). *For every $G = (V, E, \omega)$ with $\omega : E \to \mathbb{Q}_{\geq 0}$, there exists an $\varepsilon \gg 0$ such that G admits a strong ε-rounding.*

Proof. Let $m := |E|$ and $\ell := \max_\pi\{\omega(\pi) \mid \pi \text{ is a shortest path in } G\}$, i. e., ℓ is defined as the weight of the longest shortest path in G. Since ω is a rational weight function, for each edge $e_i \in E$ there exist two integer values $p_i, q_i \in \mathbb{N}_0, q_i \neq 0$ such that $\omega(e_i) = {}^{p_i}/_{q_i}$. Let N be the least common multiple of the denominators, i. e., $N := \operatorname{lcm}(q_1, \ldots, q_m)$. Multiplying the weight of each edge with N we obtain an integer-valued weight function $\widetilde{\omega}$, where $\widetilde{\omega}(e_i) := N \cdot \omega(e_i)$ for $1 \leq i \leq m$. For every path $\pi \in G$, this function implies the following identity

$$\widetilde{\omega}(\pi) = \sum_{e \in \pi} \widetilde{\omega}(e) = \sum_{e \in \pi} N \cdot \omega(e) = N \cdot \omega(\pi). \tag{4.1}$$

Thus, conditions 1 to 3 can be easily verified using Equation 4.1 and the definition of shortest paths:

1. For every shortest path $\pi \in G$, it is

$$|\,\widetilde{\omega}(\pi) - \omega(\pi)\,| \overset{(4.1)}{=} |\,N \cdot \omega(\pi) - \omega(\pi)\,| = |\,(N-1) \cdot \omega(\pi)\,| < N \cdot l.$$

2. Also every shortest path π in G is a shortest path in $\widetilde{G}$, since for every path π' we have

$$\widetilde{\omega}(\pi) \overset{(4.1)}{=} N \cdot \omega(\pi) \leq N \cdot \omega(\pi') \overset{(4.1)}{=} \widetilde{\omega}(\pi').$$

3. Finally, every shortest path π in $\widetilde{G}$ was already a shortest path in G, since

$$\omega(\pi) \overset{(4.1)}{=} \frac{1}{N} \cdot \widetilde{\omega}(\pi) \leq \frac{1}{N} \cdot \widetilde{\omega}(\pi') \overset{(4.1)}{=} \omega(\pi').$$

Thus, G admits a strong $(N \cdot \ell)$-rounding. □

The proof of this observation also shows that an identity like Equation 4.1 is sufficient to establish the existence of a strong ε-rounding; moreover, the identity seems necessary if both of the conditions 2 and 3 have to be satisfied. This suggests that a strong rounding might not always exists for an arbitrary real-valued weight function, although, this is no formal disproof.

4.1.2 Related Work

Funke and Storandt [40] consider a rounding problem on undirected graphs with the following conditions: the *absolute* change in weight of any edge should be less than one; the *relative* change in weight between every two vertices u and v whose shortest path (before rounding) consists of *at least* k edges, should be bounded. No restrictions are imposed on the change of weight of paths of fewer than k edges, regardless of their weight.

Compared to our model as introduced above, there are two major differences to Funke and Storandt's model [40]. One important difference to our model is that, even when distances are preserved, Funke and Storandt [40] allow shortest paths to change, which we forbid for weak/strong ε-roundings. The essential difference, however, is that Funke and Storandt consider relative errors rather than absolute errors: If x is the distance between u and v before rounding, and $\widetilde{x}$ is the distance between u and v after rounding, then the rounding error would be $\max({}^{x}/_{\widetilde{x}}, {}^{\widetilde{x}}/_{x})$. This might ease the

problem because of the following property of relative errors (which Funke and Storandt exploit in their heuristics): If the relative change of weight on any subpath of a path is bounded, then the relative change of weight on the complete path automatically adheres to the same bound. To prevent the rounding errors on short paths from dominating the result, Funke and Storandt introduce an input parameter k. They only require the relative rounding error to be at most $1+\varepsilon$ for distances between vertices that are at least k edges apart along the shortest path. For individual edges, they require an absolute rounding error less than 1. This implies that the rounding error of a path depends on how the path is subdivided into edges: If the path consists of m edges where $m < k$, an *absolute* rounding error close to m is accepted; if $m \geq k$, the *relative* rounding error is bounded to $1+\varepsilon$.

Funke and Storandt argue that, in general, many rounding problems are NP-hard. They describe an integer linear program for their problem that appears to be too expensive to solve even for small graphs. For this reason, they turn to describing and evaluating a greedy rounding heuristic. Note that the rounding problem as studied by Funke and Storandt may, in some sense, be easier than ours, as they can exploit a subpath property: If the relative change of weight on any subpath of a path is bounded, then the relative change of weight on the complete path automatically adheres to the same bound. For absolute changes in weight, this property does not hold.

In follow-up work, Storandt [96] draws attention to the importance of preserving actual shortest paths and not only distances. For graphs that consist of a single path, she shows how to round the weights in linear time such that the absolute change in weight of any subpath is less than one; for completeness, we will describe the solution in Section 4.3. For general graphs, Storandt proposes to augment the graphs with additional edges that can represent paths of many edges in the original graphs without suffering from accumulated rounding errors. Note, however, that this approach may be suitable if the goal is a data structure for shortest path queries; the approach is not suitable for the map drawing application that we mentioned in the introduction.

Asano et al. [5] study roundings in the following setting. The original graph A has a weight function that assigns real weights in $[0,1]$ to all *vertices*, not edges. A set $\mathscr{F}$ of paths in A is given, whose weights should be maintained. A valid rounding assigns weights in $\{0,1\}$ to all vertices of A such that the total weight of each path in $\mathscr{F}$ changes by less than one. The problem is superficially similar to ours, as we could define A to consist of weighted vertices that represent the weighted edges of our graph G, and we could define $\mathscr{F}$ as the set of shortest paths in G (not A). However, hardness results mentioned by Asano et al. do not apply to our problem because our problem is more specific. Asano et al. obtain combinatorial results for the case in which $\mathscr{F}$ consists of shortest paths in A (not G). Note that, in any case, Asano et al. do not consider a shortest path maintenance condition: their roundings do not guarantee that any shortest path in the original graph is still a shortest path in the rounded graph, as there are no restrictions on the total change of weight on paths not in $\mathscr{F}$.

Finally, note that various authors have studied roundings in the following setting. The input is a hypergraph H—to distinguish it from the graphs in our paper, we will call the vertices of H *hypervertices* and its edges *hyperedges*. The hypervertices have real weights in $[0,1]$. Each hyperedge is a set of at least two hypervertices; its weight is the sum of the weights of its hypervertices. The goal is to find a *global rounding*, that is, replace the hypervertex weights by integers such that the change of weight on each hypervertex and each hyperedge is less than one. Note that our path-oblivious 1-rounding problem can be formulated in these terms.

Asano et al. [7, 6] proved that finding a global rounding for such a hypergraph is NP-hard if the hypervertices represent cells of a square grid and the hyperedges represent squares of 2×2 cells. Later, Asano with different co-authors [5], studied the rounding problem as described above; in the notion of hypergraphs, it can be stated as follows: The hypervertices represent the n vertices of a

graph A with weights on vertices; the hyperedges represent all shortest paths in A. They conjectured that in this case, at most $n+1$ global roundings of the (hyper-)vertex weights are possible. This was proven for path-shaped graphs [89], and later also for outerplanar graphs [97]. However, to establish a relation to our (path-oblivious, weak, or strong) ε-rounding problem, one would have to reduce the square grid rounding problem to our problem; or, our problem of finding a rounding of the edge weights to the problem of finding a rounding of the (hyper-)vertex weights.

4.2 Complexity of the Problem

By reduction from 3-SAT, we prove that it is NP-hard to decide, given an edge-weighted graph G and an error tolerance $\varepsilon \in (3/4; 1]$, whether G admits a path-oblivious, weak, or strong ε-rounding. The reduction we present, proves hardness for all three variants of the problem.

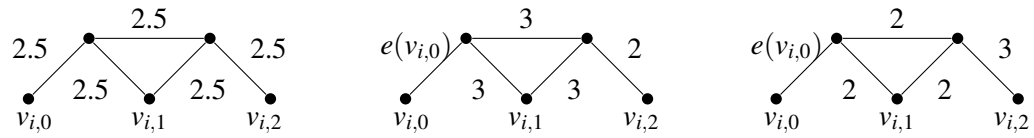
The 3-SAT problem is the following. Given a 3-CNF formula, i. e., a Boolean formula α in conjunctive normal form, where each of the m clauses consists of exactly three literals. Each literal is either one of n variables $x_1, x_2, \ldots, x_n$ or its negation. Decide whether α is satisfiable. W. l. o. g. we assume that every variable appears at most once in each clause of the 3-SAT formula.[2]

In the following, we show how to construct a graph $G_\alpha = (V, E, \omega)$ for a given 3-CNF formula α such that G_α admits a strong ε-rounding if α is satisfiable, whereas G_α does not even admit a path-oblivious ε-rounding if α is not satisfiable. To describe G_α, we introduce subgraphs called *variable gadgets* and *clause gadgets*, as well as *clause-variable edges* and *shortcut edges*.

The idea of the construction is as follows and works for every $\varepsilon \in (3/4; 1]$. In Lemma 12, we will show that a variable gadget admits exactly two strong ε-roundings. We identify these two roundings with the assignments `true` and `false` of a Boolean variable. Using clause-variable edges, the state of a variable gadget can be transferred to a clause gadget (Lemma 14). Locally, the clause gadget admits an ε-rounding if and only if one of the variable assignments (transferred via clause-variable edges) satisfies the clause (Lemma 13). We use shortcut edges to ensure that shortest paths in G_α which do not contribute to modelling α, are easy to analyse and unique—before and after rounding the weights (Lemma 15).

To design a *variable gadget* first consider two edges attached to a triangle, where each edge is of weight 2.5, as illustrated in Figure 4.1i. In an ε-rounding, the choice of the rounding for $e(v_{i,0})$, the edge incident on $v_{i,0}$, determines the rounding of the remaining edges; see Figure 4.1ii and Figure 4.1iii. To obtain a variable gadget for variable x_i, we proceed as follows. Assume that x_i appears in h literals $l_1, \ldots, l_h$ of α. We construct h triangles $\Delta_{i,1}, \ldots, \Delta_{i,h}$, where each $\Delta_{i,k}$ (for $k \in \{1, ..., h\}$) has a left vertex, a right vertex, and a *base* (bottom) vertex; we label the base vertex $v_{i,k}$. We chain up the triangles by including an edge between the right vertex of $\Delta_{i,k}$ and the left vertex of $\Delta_{i,k+1}$ for each $k \in \{1, ..., h-1\}$. To the left vertex of $\Delta_{i,1}$, we attach another vertex $v_{i,0}$, and to the right vertex of $\Delta_{i,h}$, we attach another vertex $v_{i,h+1}$, as shown in Figure 4.2. Finally, for every $1 \leq k \leq h$, if $l_k = \neg x_i$, we add another vertex $\overline{v}_{i,k}$, called *inverter*, which we connect to $v_{i,k}$. All edges of the variable gadget have an initial weight of 2.5. We call the edges of $\Delta_{i,1}, \ldots, \Delta_{i,h}$ *triangle edges*. Moreover, with $e(v_{i,0})$ we denote the unique edge of the variable gadget attached to $v_{i,0}$.

[2] Otherwise, α can be transformed in polynomial time as follows: First, remove all clauses that contain a variable and its negation, since they are always satisfied. Next, introduce three new variables a, b, and c. In every clause that contains two or three copies of the same literal, replace the second copy by b and the third copy (if it exists) by c. Finally, append the following clauses to α: $(\neg a \vee \neg b \vee \neg c), (\neg a \vee \neg b \vee c), (\neg a \vee b \vee \neg c), (a \vee \neg b \vee \neg c), (a \vee \neg b \vee c), (a \vee b \vee \neg c)$. The additional clauses ensure that b and c are `false` in every satisfying assignment. Therefore, they do not affect the original clauses in which they replace a duplicate literal.



(i) The initial weights. (ii) $e(v_{i,0})$ rounded to 2. (iii) $e(v_{i,0})$ rounded to 3.

Figure 4.1: A minimal variable gadget. In an ε-rounding of this gadget for $\varepsilon \in (1/2; 1]$, every shortest path of two edges has to round one edge up and the other one down; otherwise the total rounding error on the path would be ± 1, violating Condition 1 of an ε-rounding. Thus, the top triangle edge must be rounded in the opposite way as compared to the edges $e(v_{i,0})$ and $e(v_{i,2})$, incident on $v_{i,0}$ and $v_{i,2}$, respectively. These arguments imply that $e(v_{i,0})$ and $e(v_{i,2})$ have to be rounded in the same way and *all* triangle edges are rounded in the opposite way. Note that paths containing two triangle edges have rounding error ± 1; however, such paths are never shortest paths, neither before nor after rounding.

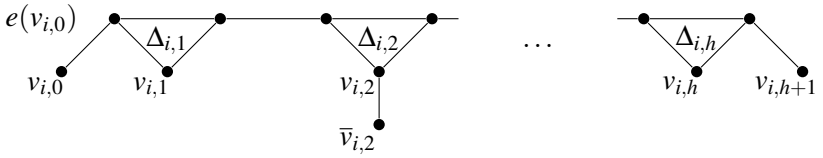


Figure 4.2: The variable gadget for x_i, where x_i appears in h literals $l_1, \ldots, l_h$ of α. In this sketch we assume $l_2 = \neg x_i$, so an additional vertex $\overline{v}_{i,2}$ is added and attached to $v_{i,2}$. All edges have weight 2.5, so the choice of the rounding for $e(v_{i,0})$ determines the rounding of all other edges in an ε-rounding for $\varepsilon \in (1/2; 1]$: triangle edges have to be rounded complementary to non-triangle edges.

A variable gadget has only two different ε-roundings for $\varepsilon \in (1/2; 1]$; moreover, in such an ε-rounding, the choice of the rounding for $e(v_{i,0})$ determines the rounding of all other edges.

Lemma 12 (ε-roundings of variable gadgets). *A variable gadget admits exactly two ε-roundings for $\varepsilon \in (1/2; 1]$ (both of which are strong ε-roundings): either all triangle edges are rounded up and all other edges down, or vice versa.*

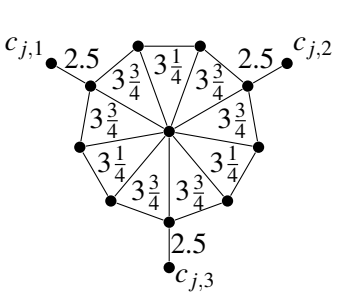
Moreover, for any two vertices u, v of the gadget, the rounding error of the unique shortest path from u to v is either zero or equal to the rounding error on the last edge of the path ending at v.

Proof. For a single triangle and its adjacent non-triangle edges, Figure 4.1 explains that all triangle edges must be rounded in the same way, while non-triangle edges must be rounded in the opposite way. Tracking the triangles in Figure 4.2 from left to right, the same holds for any number of triangles, by induction.

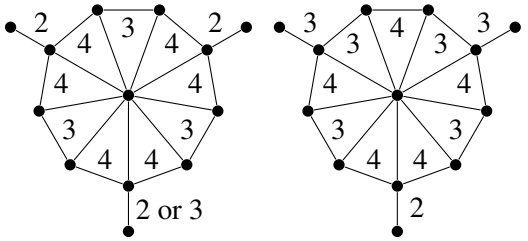
We will now argue that these two roundings satisfy the conditions of a strong ε-rounding. Note that, before and after rounding, paths with two consecutive triangle edges are no shortest paths. Thus, a simple path in this gadget is a shortest path before rounding if and only if it is a shortest path after rounding, and in any shortest path triangle and non-triangle edges alternate. Therefore, any shortest path with an even number of edges has rounding error zero; any shortest path with an odd number of edges has the rounding error of its last edge. This establishes Conditions 1, 2 and 3 of a strong ε-rounding and thus completes the proof. □

To create a *clause gadget* for clause C_j, we take a cycle of nine vertices and nine edges where each edge gets an initial weight of 3.6. Moreover, we attach, to every third vertex along the cycle, a

new vertex, called a *knob*, with another edge of weight 2.5, called a *handle*. We denote the knobs by $c_{j,1}, c_{j,2}, c_{j,3}$, as shown in Figure 4.3i. We will use the notation $e(c_{j,t})$ to denote the edge (handle) of the clause gadget that connects $c_{j,t}$ to the cycle. Finally, we add a vertex which we connect to every vertex on the cycle with an edge of weight 6. Note that the weights of these edges are integer; hence, they cannot be rounded. A clause gadget has at least three strong 1-roundings, see Figure 4.3ii. However, as the following lemma shows, there is no path-oblivious, weak, or strong ε-rounding for the clause gadget in which $e(c_{j,1})$, $e(c_{j,2})$ and $e(c_{j,3})$ are all rounded up.



(i) The initial weight of all edges incident on the centre is 6.



(ii) Three ε-roundings of the clause gadget. No two consecutive edges along the nonagon can be rounded down in an ε-rounding. At most two of the edges $e(c_{j,1}), e(c_{j,2})$ and $e(c_{j,3})$ are rounded up.

Figure 4.3: The clause gadget for clause C_j and $\varepsilon \in (3/4; 1]$.

Lemma 13 (ε-roundings of a clause gadget). *Consider a clause gadget for C_j and its three handles $e(c_{j,1}), e(c_{j,2})$ and $e(c_{j,3})$.*

The clause gadget admits a path-oblivious ε-rounding for $\varepsilon \in (3/4; 1]$ if and only if at least one of its three handles is rounded down. If there is a path-oblivious ε-rounding, there is a strong ε-rounding.

Proof. Figure 4.3ii shows a strong ε-rounding for the clause gadget for the cases in which one, two, or three of the edges $e(c_{j,1}), e(c_{j,2})$ and $e(c_{j,3})$ are rounded down. It remains to show that if none of these three edges are rounded down, it is impossible to obtain an ε-rounding (not even a path-oblivious one) for the complete gadget.

Assume, to the contrary, that the weights of $e(c_{j,1})$, $e(c_{j,2})$, and $e(c_{j,3})$ are all rounded up. For the shortest path between any pair of $c_{j,1}, c_{j,2}, c_{j,3}$, we now know that (at least) two of the three edges along the nonagon have to be rounded down. Otherwise, the shortest path of length $2 \cdot 2.5 + 2 \cdot 3.75 + 3.25 = 15.75$ w. r. t. ω would have weight (at least) $2 \cdot 3 + 1 \cdot 3 + 2 \cdot 4 = 17$ w. r. t. $\widetilde{\omega}$, contradicting Condition 1 of an ε-rounding. However, if at least six of the nine edges along the nonagon have to be rounded down, there have to be two adjacent edges $\{u, v\}, \{v, w\}$ along the nonagon, which are rounded down. Consequently, the shortest path from u to w has weight at least 7 w. r. t. ω but at most 6 w. r. t. $\widetilde{\omega}$, which again contradicts Condition 1 of an ε-rounding. □

Next, we introduce *clause-variable edges* to connect the variable gadgets to the clause gadgets containing these variables. More precisely, if variable x_i appears in clause C_j, we connect the corresponding gadgets with exactly one clause-variable edge of weight $D := 5m + 20$ according to the following rule: if the t-th literal in C_j is x_i, then we connect a base vertex of the variable gadget for x_i to $c_{j,t}$, using an edge of weight D; if the t-th literal in C_j is $\neg x_i$, then we connect an inverter of the variable gadget for x_i to $c_{j,t}$, using an edge of weight D. We do this such that exactly one clause-variable edge is connected to each inverter, and exactly one clause-variable edge is connected

to each base vertex that is not attached to an inverter. By design, the variable gadgets have the right numbers of base vertices and inverters to make this possible.

Note that clause-variable edges do not invalidate Lemmata 12 and 13. They still hold with respect to the shortest paths between any pair of vertices of the variable or clause gadget, respectively. This can be seen as follows. There are m clauses and each variable appears in each clause at most once. Hence, the diameter of a variable gadget is at most $(2m+1)\cdot 2.5$. The diameter of a clause gadget is 15.75. Thus, the variable and clause gadgets all have diameter less than $D-2$. Therefore, before rounding, no path between two vertices of the same gadget that uses a clause-variable edge can be a shortest path. Moreover, when we choose an ε-rounding for each gadget separately, the rounded weight of the shortest path between any pair of vertices within a gadget will still be less than $D-1$, while the rounded weight of any path using a clause-variable edge will still be at least D. Therefore, also after rounding, no path between two vertices of the same gadget that uses a clause-variable edge can be a shortest path. Thus, adding the clause-variable edges does not invalidate Lemmas 12 and 13.

Further note that due to this construction, the choice for $e(v_{i,0})$ also determines the rounding for $e(c_{j,t})$ in an ε-rounding:

Lemma 14 (clause-variable edges and ε-roundings). *For any ε-rounding on G_α with $\varepsilon \in (3/4; 1]$:*

- *If $c_{j,t}$ is connected to a base vertex of the variable gadget for x_i, then $e(c_{j,t})$ is rounded in the same way as $e(v_{i,0})$;*
- *if $c_{j,t}$ is connected to an inverter vertex of the variable gadget for x_i, then $e(c_{j,t})$ is rounded in the opposite way as $e(v_{i,0})$.*

Proof. Consider the t-th literal l of clause C_j ($t \in \{1,2,3\}$).

If $l = x_i$, then a clause-variable edge connects $c_{j,t}$ to a base vertex of the variable gadget for x_i. Now consider the path that consists of a triangle edge incident to this base vertex (with weight 2.5), the clause-variable edge (with integer weight D), and $e(c_{j,t})$ (with weight 2.5). This path has length $D+5$ and is a shortest path: Any other path between the same vertices would have to make a detour in the variable gadget and has length at least $D+7.5$; or, it leads over at least three other clause-variable edges and has length at least $3D$. Because the path has integer length w. r. t. ω, it cannot change its weight in any ε-rounding. It follows that $e(c_{j,t})$ must be rounded in the opposite way as compared to the triangle edge in the variable gadget, which, by Lemma 12, implies that $e(c_{j,t})$ is rounded like the non-triangle edge $e(v_{i,0})$.

Otherwise, $l = \neg x_i$ and a clause-variable edge connects $c_{j,t}$ to an inverter. Now we consider the path that consists of the non-triangle edge incident on the inverter in the variable gadget, the clause-variable edge, and $e(c_{j,t})$. Again, the last edge must be rounded in the opposite way as compared to the first, which, by Lemma 12, is rounded in the same way as $e(v_{i,0})$. □

Finally, to ensure that the shortest path between any pair of vertices of G_α is unique (and easy to analyse), we add *shortcut edges* according to the following rule. If u and v are vertices of G_α such that one of the following conditions holds:

(i) u and v belong to different variable gadgets;

(ii) u and v belong to different clause gadgets;

(iii) u belongs to a variable gadget for variable x_i and v belongs to a clause gadget for clause C_j and neither x_i nor $\neg x_i$ appears in C_j;

then we include an edge $\{u,v\}$ in G_α with weight $2D$.

Lemma 15 (shortest path via shortcut edge). *Let u and v be vertices of G_α that are directly connected by a shortcut edge, and let $\widetilde{\omega}$ be an ε-rounding with $\varepsilon \in (3/4;1]$ on G_α. Then, the shortcut edge $\{u,v\}$ is the unique shortest path in G_α with respect to ω and $\widetilde{\omega}$.*

Proof. By construction, u and v belong to different gadgets and there is no clause-variable edge between these gadgets. Therefore, any path from u to v other than the direct shortcut edge $\{u,v\}$, must use either (a) another shortcut edge of weight $2D$ plus at least one other edge of weight at least 2.5 or (b) at least two clause-variable edges of weight D each, plus at least one other edge of weight at least 2.5 (because no vertex is incident on more than one clause-variable edge). In both cases, before and after rounding, the total weight of the path would be at least 2 more than the weight of the shortcut edge $\{u,v\}$, which is $2D$. Therefore, $\{u,v\}$ is the unique shortest path in G_α with respect to both, ω and $\widetilde{\omega}$. □

Note that, just like clause-variable edges, the shortcut edges do not invalidate Lemmas 12 and 13. They do not invalidate Lemma 14 either, as its proof hinges on shortest paths of length $D+5 < 2D-2$. Figure 4.4 exemplifies the construction of G_α.

We can finally prove the following theorem.

Theorem 9. *Given an edge-weighted graph G and an error tolerance ε, it is NP-hard to decide whether G admits (1) a path-oblivious ε-rounding; (2) a weak ε-rounding; (3) a strong ε-rounding.*

The proof assumes that $\varepsilon \in (3/4;1]$ and can be sketched as follows. On the one hand, we show how to obtain a strong ε-rounding for G_α if α has a satisfying assignment: For each variable x_i, we round $e(v_{i,0})$ down if the assignment sets x_i to `true`; otherwise, we round $e(v_{i,0})$ up. We use Lemmata 12 to 14 to make it a strong ε-rounding on G_α. On the other hand, we show that if G_α admits a path-oblivious ε-rounding, then α is satisfiable. This we do by constructing a satisfying assignment for α from a given path-oblivious ε-rounding: If the weight of $e(v_{i,0})$ is rounded down, we set x_i to `true`, otherwise to `false`.

Proof of Theorem 9. For the proof we assume that $\varepsilon \in (3/4;1]$ and show that if α is satisfiable, G_α admits a strong ε-rounding. Moreover, if G_α admits a path-oblivious ε-rounding, then α is satisfiable.

To start with, we show how to obtain a strong ε-rounding for G_α if α is satisfiable. Let ψ be an assignment of values to the variables that satisfies α. So we have $\psi : \{x_1,\ldots,x_n\} \to \{0,1\}$, where 0 denotes the logical value `false` and 1 denotes `true`; correspondingly, $\psi(\neg x_i) = 1 - \psi(x_i)$. Then, a strong ε-rounding $\widetilde{\omega}$ for G_α can be constructed in the following way:

- edges with integer weight keep their weight;
- in a variable gadget for x_i: if $\psi(x_i) = 0$, all triangle edges are rounded down and all non-triangle edges (including $e(v_{i,0})$) are rounded up; if $\psi(x_i) = 1$, all triangle edges are rounded up and all non-triangle edges (including $e(v_{i,0})$) are rounded down;
- in a clause gadget for clause C_j with literals l_t for $t \in \{1,2,3\}$: If $\psi(l_t) = 0$, the weight of $e(c_{j,t})$ is rounded up; if $\psi(l_t) = 1$, the weight of $e(c_{j,t})$ is rounded down. Observe that this implies the following: If $l_t = x_i$, then $e(c_{j,t})$ is rounded like the non-triangle edges in the variable gadget for x_i; if $l_t = \neg x_i$, then $e(c_{j,t})$ is rounded like the triangle edges. Since α is satisfied, also C_j is satisfied and at least one of the edges $e(c_{j,1})$, $e(c_{j,2})$, $e(c_{j,3})$ is rounded down. By Lemma 13, we can complete the rounding of the gadget to one of those given in Figure 4.3ii (modulo

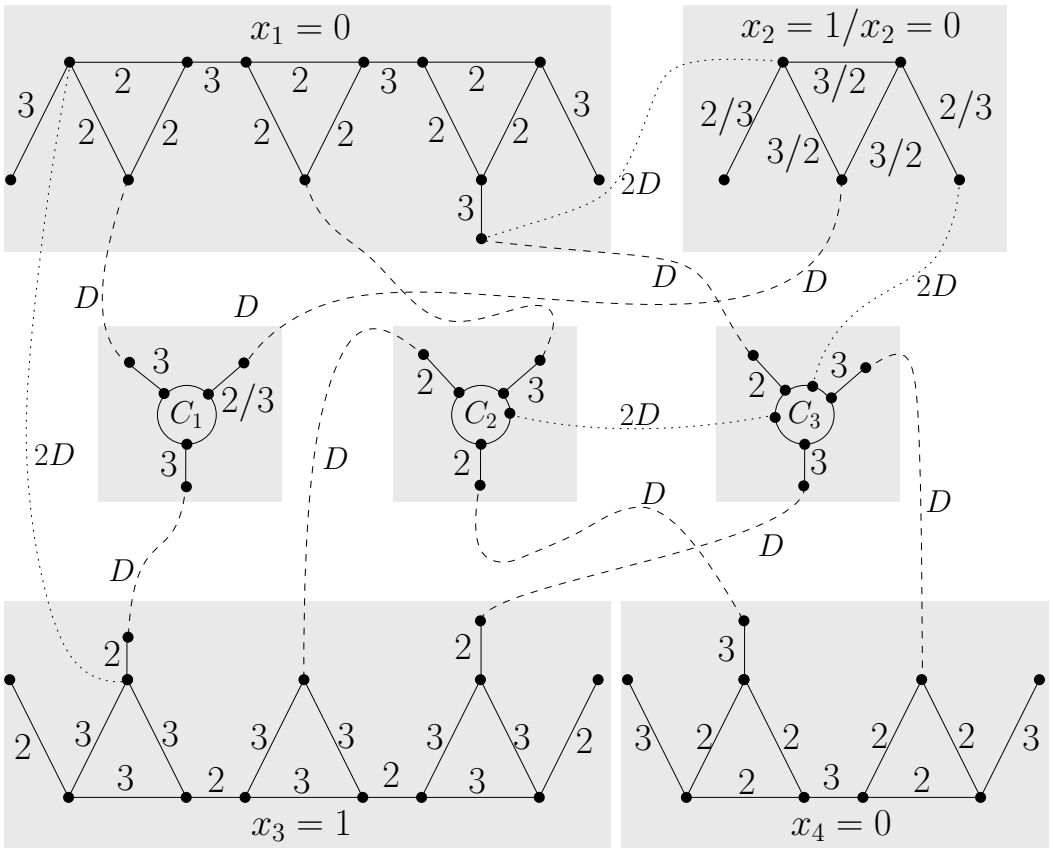


Figure 4.4: A sketch of G_α for $\alpha = (x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee x_3 \vee \neg x_4) \wedge (\neg x_1 \vee \neg x_3 \vee x_4)$. Grey bounding boxes mark gadgets, clause-variable edges are dashed and have weight $D = 5 \cdot 3 + 20 = 35$, shortcut edges (mostly omitted) are dotted and have weight $2D$. The weights of the edges in G_α have been rounded according to the assignment of the variables, which are given in the boxes of the corresponding variable gadgets. Both assignments are given for x_2, which affects the rounding of several edges: The rounded weight of each affected edge is given by a/b where a corresponds to the assignment $x_2 = 1$ and b corresponds to $x_2 = 0$. Whether one can obtain an ε-rounding for $\varepsilon \in (3/4; 1]$ depends of the choice for x_2: If $x_2 = 1$, we can obtain an ε-rounding for the clause gadget for C_1 and G_α; whereas for $x_2 = 0$, this is already impossible for the clause gadget for C_1, since all of its handles are rounded up.

rotation). That is, for each pair of edges from $e(c_{j,1}), e(c_{j,2})$, and $e(c_{j,3})$, if both of them are rounded up, we round the three edges between them along the nonagon to 3, 4, 3; in all other cases, we round the three edges between them along the nonagon to 4, 3, 4.

Certainly, the weight of every edge has been rounded either up or down and is now integer. It remains to prove that Conditions 1, 2 and 3 of a strong ε-rounding are fulfilled for the shortest paths between any pair of vertices u and v in G_α. Let u, v be vertices of G_α. If G_α contains a shortcut edge $\{u,v\}$, Lemma 15 applies and the conditions hold. Otherwise, u, v are not connected by a shortcut edge and we have to distinguish two cases:

(i) u and v lie in the same gadget;

(ii) u lies in a variable gadget for a variable x_i and v in a clause gadget for a clause C_j and either x_i or $\neg x_i$ appears in C_j.

(i) For each gadget, we constructed a strong ε-rounding according to Lemma 12 or 13. As observed above, the lemmata continue to hold after adding clause-variable edges and shortcut edges. Thus, if u and v lie in the same gadget, Conditions 1, 2, 3 are satisfied with respect to shortest paths between u and v.

(ii) Since each variable occurs at most once in any clause, the gadgets of u and v are, by construction, connected by exactly one clause-variable edge $\{s,c\}$, where s is a base or inverter vertex in the variable gadget, and $c = c_{j,t}$ is a handle of the clause gadget, for some t.

We start with verifying Condition 1 of the ε-rounding for a shortest path $\pi(u,v)$ from u to v that uses the clause-variable edge $\{s,c\}$. For any two vertices a and b on $\pi(u,v)$, let $\pi(a,b)$ be the subpath of $\pi(u,v)$ from a to b. If C_j includes the positive literal x_i, then s is a base vertex of the variable gadget, by construction. Consider the unique shortest path $\pi(u,s)$ from u to s in the variable gadget. By Lemma 12, the rounding error on $\pi(u,s)$ is either zero or equal to the rounding error on any triangle edge (which is either -0.5 or 0.5). Adding $\{s,c\}$ to the path does not change the rounding error, as it has integer weight. If $v = c$, we are done now. Otherwise, the subpath $\pi(c,v)$ from c to v in the clause gadget uses the edge $e(c_{j,t}) = \{c,c'\}$. By Lemma 14, $e(c_{j,t})$ is rounded in the same way as the non-triangle edges in the variable gadget. It follows that the rounding error of either $\pi(u,c)$ or $\pi(u,c')$ is zero. The remaining part of the path from u to v, that is $\pi(c,v)$ or $\pi(c',v)$, lies entirely inside the clause gadget and is the unique shortest path to v from c or c', respectively, with rounding error more than $-\varepsilon$ and less than ε, by Lemma 13. Hence, Condition 1 is satisfied with respect to $\pi(u,v)$. If C_j includes the negative literal $\neg x_i$, then s is an inverter vertex, and the whole argument goes through with the roles of triangle and non-triangle edges swapped.

The weight of $\pi(u,v)$ w. r. t. ω is bounded by $\omega(\pi(u,v)) = \omega(\pi(u,s)) + D + \omega(\pi(c,v)) \leq 5m + D + 15.8 = 2D - 4.2 \leq 3D - 2$. Since Condition 1 is satisfied, the weight with respect to $\widetilde{\omega}$ is therefore less than $3D-1$. This bound rules out other paths that do not use $\{s,c\}$: any other path from u to v that avoids the clause-variable edge $\{s,c\}$ must pass by (a) a single other gadget, or (b) at least two other gadgets. In case (a), the path must use at least two clause-variable edges or shortcut edges, each of weight at least D. In fact, at least one of these edges must be a shortcut edge of weight $2D$, because a path from u with two clause-variable edges and no shortcut edges, could only end in a variable gadget. In case (b), the path uses at least three clause-variable edges or shortcut edges, each of weight at least D. In both cases (a) and (b), the total weight of the path would be at least $3D$; however, such a path cannot be shortest, neither before nor after rounding.

Since the shortest paths within the gadgets are maintained by Lemma 12 and 13, it follows that also Conditions 2 and 3 are satisfied with respect to u and v; i. e., a shortest path between u and v w. r. t. ω is shortest w. r. t. $\widetilde{\omega}$ and vice versa.

This completes the proof that if α is satisfiable, then G_α admits a strong ε-rounding. It remains to prove that if G_α admits a path-oblivious ε-rounding, then α is satisfiable. This we do by constructing, from a given path-oblivious ε-rounding, a choice ψ of the variables that satisfies α.

Assume we are given an ε-rounding $\widetilde{\omega}$ of G_α. If in $\widetilde{\omega}$, the weight of $e(v_{i,0})$ is rounded down, we set $\psi(x_i) = 1$, otherwise we set $\psi(x_i) = 0$. Now consider the clause gadget for any clause C_j. Following Lemma 13, we know that there is at least one $t \in \{1,2,3\}$ such that $e(c_{j,t})$ is rounded down. By construction, a clause-variable edge connects $c_{j,t}$ to a base vertex or an inverter of the variable gadget for a variable x_i: a base vertex if x_i appears as a literal in C_j, and an inverter if $\neg x_i$ appears as a literal in C_j. Following Lemma 14, in the first case $e(v_{i,0})$ is rounded in the same way as $e(c_{j,t})$, that is, down, and thus $\psi(x_i) = 1$ and the literal x_i makes C_j true. In the second case, $e(v_{i,0})$ is rounded in the opposite way of $e(c_{j,t})$, that is, up, and thus $\psi(x_i) = 0$, and the literal $\neg x_i$ makes C_j true. The same argument applies to each clause C_j, and thus, ψ satisfies α.

To complete the proof, observe that 3-SAT is an NP-hard problem and G_α consists of $O(mn)$ vertices and $O(m^2n^2)$ edges and can be constructed in polynomial time. □

4.3 A Quadratic-Time Algorithm for Trees

In the following, we present algorithms for the special case in which the graph is a tree. Note that in this case there is only one simple path between any pair of vertices; hence, there is no difference between path-oblivious, weak, and strong ε-roundings.

Clearly, if the whole graph is a path with edges $e_1, ..., e_n$, a 1-rounding always exists; moreover, it can be computed in linear time, assuming the floor function can be computed in constant time, see [89, 96]: Let d_i be $1/2 + \sum_{j=1}^{i} \omega(e_i)$; then we set $\widetilde{\omega}(e_i) = \lfloor d_i \rfloor - \lfloor d_{i-1} \rfloor$. Now, for any subpath $e_a, ..., e_z$, we have $\sum_{i=a}^{z} \widetilde{\omega}(e_i) = \lfloor d_z \rfloor - \lfloor d_{a-1} \rfloor < d_z - (d_{a-1} - 1) = 1 + \sum_{i=a}^{z} \omega(e_i)$ and $\sum_{i=a}^{z} \widetilde{\omega}(e_i) = \lfloor d_z \rfloor - \lfloor d_{a-1} \rfloor > (d_z - 1) - d_{a-1} = -1 + \sum_{i=a}^{z} \omega(e_i)$; thus $\widetilde{\omega}$ satisfies Condition 1 for $\varepsilon = 1$, and $\widetilde{\omega}$ is a 1-rounding. Sadakane et al. [89] prove that a path of n vertices admits at most n different 1-roundings, describe how to compute all 1-roundings in $O(n^2)$ time, and show how to determine a 1-rounding with the smallest maximum absolute rounding error in the same time.

If the graph is a tree, observe that we can obtain a 2-rounding in linear time as follows. Choose any vertex of the tree as the root r. For any other vertex u, let $p(u)$ be the parent of u, and let d_u be the (unrounded) weight of the path from r to u. Now we set $\widetilde{\omega}(\{p(u), u\}) = \lfloor d_u \rfloor - \lfloor d_{p(u)} \rfloor$. By the same calculation as above, for any vertex u, the absolute rounding error $|e(u,v)|$ on any path from u to an ancestor v of u is now less than one. Now, given two arbitrary vertices u and w, let v be their lowest common ancestor. The absolute rounding error on the path from u to w is at most $|e(u,v)| + |e(v,w)| < 2$.

We now present an algorithm that decides, given a tree T and an error threshold $\varepsilon < 2$, in quadratic time, whether T admits an ε-rounding. We choose an arbitrary vertex of T as the root r. We say v is a descendant of u if u lies on the path from r to v. For any vertex u, the subtree T_u of T is the subgraph of T that is induced by all descendants of u; this vertex u is called the root of T_u, see Figure 4.5. By $|T|$ we denote the number of vertices of T; by $\pi(u,v)$ we denote the path in T from u to v.

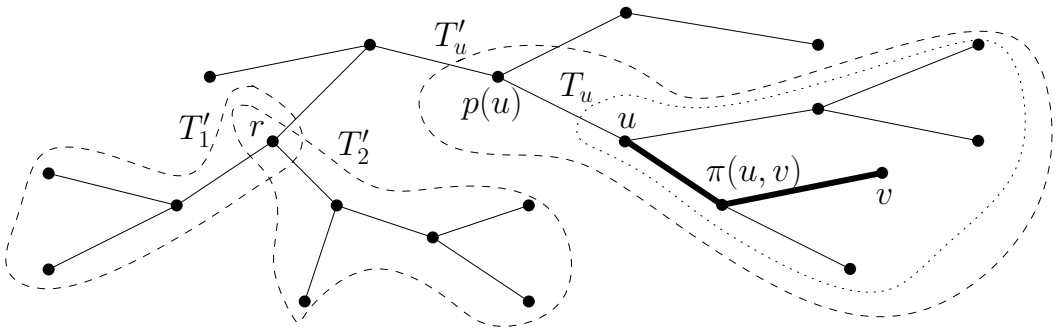


Figure 4.5: A rooted tree T with root vertex r. The subtree T_u with root u can be extended to a subgraph T'_u with root $p(u)$ by adding the edge $\{u, p(u)\}$.

Definition 11 (root error range). *Let $\widetilde{\omega}$ be an ε-rounding on a tree T with root r. For any $v \in T$, let $e(r,v)$ be the rounding error on $\pi(r,v)$, that is, $e(r,v) \coloneqq \widetilde{\omega}(\pi(r,v)) - \omega(\pi(r,v))$.*

We call the smallest interval that contains the signed rounding errors of the paths from r to all vertices of T the root error range $E(T,\widetilde{\omega})$*, so*

$$E(T,\widetilde{\omega}) \coloneqq \left[\min_{v\in T} e(r,v), \max_{v\in T} e(r,v)\right].$$

Note that $e(r,r) = 0$, so if T is a leaf, then $E(T,\widetilde{\omega}) = [0,0]$.

We call a rounding $\widetilde{\omega}$ of T locally optimal *if there is no other rounding $\widetilde{\omega}'$ of T such that the corresponding root error range $E(T,\widetilde{\omega}')$ is smaller than and contained in $E(T,\widetilde{\omega})$.*

Let the error range set $\mathscr{E}(T)$ *be the set of root error ranges that can be realized by locally optimal ε-roundings of T, that is, the set $\mathscr{E}(T) \coloneqq \{E(T,\widetilde{\omega}) \mid \widetilde{\omega}$ is a locally optimal rounding$\}$.*

Lemma 16 (error range set size). *$\mathscr{E}(T)$ has at most $2|T|$ elements.*

Proof. Observe that in any ε-rounding with $\varepsilon < 2$, the weight of any path $\pi(r,v)$ is rounded to $\lfloor\omega(\pi(r,v))\rfloor - 1$, $\lfloor\omega(\pi(r,v))\rfloor$, $\lceil\omega(\pi(r,v))\rceil$, or $\lceil\omega(\pi(r,v))\rceil + 1$.

Any root error range $E(T,\widetilde{\omega})$ includes 0, since $e(r,r) = 0$. Therefore, the lower bound of any root error range $E(T,\widetilde{\omega})$ is zero or negative, and it must be the rounding error on some path $\pi(r,v)$ whose rounded weight is $\lfloor\omega(\pi(r,v))\rfloor$ or $\lfloor\omega(\pi(r,v))\rfloor - 1$. Since v must be one of the n vertices of T, this implies that there are at most $2n$ possible values for the lower bound of any root error range.

Because $\mathscr{E}(T)$ contains only root error ranges of locally optimal ε-roundings, no two elements of $\mathscr{E}(T)$ can have the same lower bound, so the total number of elements of $\mathscr{E}(T)$ is also bounded by $2n$. □

Our algorithm computes the error range set for every subtree of T bottom-up. For this purpose, we need two subalgorithms. The first subalgorithm (explained in the proof of Lemma 17) adds, to a given subtree, the edge that connects the root to its parent in T. The second subalgorithm (explained in the proof of Lemma 18) combines two such parent-added subtrees who have a common parent. In the description of these algorithms, we assume that error range sets are sorted in ascending order by the lower bounds of the root error ranges. Since error range sets contain only root error ranges of locally optimal roundings, no element of an error range set can be contained in another. Therefore, the fact that the error range sets are sorted by ascending lower bound, implies that they are also sorted by ascending upper bound.

Let T'_u be the subgraph of T that consists of T_u and the edge between u and its parent $p(u)$ in T; we choose $p(u)$ as the root of T'_u, see Figure 4.5.

Lemma 17 (moving up). *Given $\mathscr{E}(T_u)$, we can compute $\mathscr{E}(T'_u)$ in $O(|T_u|)$ time.*

Proof. Let f denote the fractional part of the weight of $\{p(u),u\}$; more precisely, $f := \omega(\{p(u),u\}) - \lfloor \omega(\{p(u),u\}) \rfloor$. Any ε-rounding for T'_u must consist of an ε-rounding for T_u combined with setting $\widetilde{\omega}(\{p(u),u\})$ to $\lfloor \omega(\{p(u),u)\} \rfloor + k$ for some $k \in \{-1,0,1,2\}$ (because $\varepsilon < 2$, no other values for $\widetilde{\omega}(\{p(u),u\})$ are allowed). For any vertex $v \in T'_u$, other than the root $p(u)$, we have $e(p(u),v) = e(p(u),u) + e(u,v) = k - f + e(u,v)$; for the root $p(u)$ we have $e(p(u),p(u)) = 0$. Thus, a choice of an ε-rounding for T_u with root error range $[a,b] \in \mathscr{E}(T_u)$, together with a choice of $k \in \{-1,0,1,2\}$, results in a rounding for T'_u whose root error range is the smallest interval that includes $[a+k-f, b+k-f]$ and 0, that is, the root error range for T'_u is $[\min(a+k-f,0), \max(0,b+k-f)]$. This rounding is an ε-rounding if and only if $-\varepsilon < a+k-f$ and $b+k-f < \varepsilon$.

Thus, the elements of $\mathscr{E}(T'_u)$ are all from the set:

$$S = \left\{ [\min(a+k-f,0), \max(0,b+k-f)] \;\middle|\; \begin{array}{c} [a,b] \in \mathscr{E}(T_u), \\ k \in \{-1,0,1,2\}, \\ -\varepsilon < a+k-f, \\ b+k-f < \varepsilon \end{array} \right\}.$$

We can compute S in lexicographical order by first computing, for each $k \in \{-1,0,1,2\}$, the set $S_k := \{[\min(a+k-f,0), \max(0,b+k-f)] \mid [a,b] \in \mathscr{E}(T_u), -\varepsilon < a+k-f, b+k-f < \varepsilon\}$ in lexicographical order from $\mathscr{E}(T_u)$, and then merging the sets S_{-1}, S_0, S_1 and S_2 into one lexicographically ordered set S.

To obtain $\mathscr{E}(T'_u)$ from S, all that remains to do is to filter out the root error ranges that are not locally optimal. To do so, we start with an empty stack and scan S in lexicographical order. When we scan an element $[a,b]$ of S, we pop elements from the stack until the stack is empty or until the top element $[a',b']$ satisfies $b' < b$; then, if $a' \neq a$, we push $[a,b]$ onto the stack. After all elements of S have been scanned, the stack contains $\mathscr{E}(T'_u)$.

Procedure FILTER describes the filtering algorithm in pseudocode; Procedure COMPUTEERRORRANGESETWITHPARENTEDGE gives the full algorithm to compute $\mathscr{E}(T'_u)$ from $\mathscr{E}(T_u)$.

Algorithm 5: FILTER($\mathscr{E}$)

Input: An set $\mathscr{E}$ of root error ranges in lexicographical order.
Output: Maximal subset $\mathscr{E}'$ of $\mathscr{E}$ such that no element of $\mathscr{E}'$ is contained in another.

```
E' ← list with sentinel element [−∞, −∞]
foreach [a,b] in E do
    [a',b'] ← last element of E'
    while b' ≥ b do Remove last element from E'; [a',b'] ← last element of E'
    if a' ≠ a then Append [a,b] to E'
return E' without the first element (sentinel)
```

Clearly, the filtering algorithm runs in linear time: for each element $[a,b]$ of S we spend time proportional to the number of pushes (appends) and pops (removes), and each element is pushed at most once and popped at most once. The correctness follows from two observations. First, we maintain the invariant that the root error ranges on the stack, from bottom to top, are sorted in

Algorithm 6: COMPUTEERRORRANGESETWITHPARENTEDGE($\mathscr{E}(T_u)$)

Input: $\mathscr{E}(T_u)$ in ascending order, where T_u has root u.
Output: $\mathscr{E}(T'_u)$ in ascending order, where $T'_u = T_u \cup \{\{p(u),u\}\}$ with root $p(u)$.

$f \leftarrow \omega(\{p(u),u\}) - \lfloor \omega(\{p(u),u\}) \rfloor$
for $k \leftarrow -1$ **to** 2 **do**
  $S_k \leftarrow$ empty list
  foreach $[a,b]$ in $\mathscr{E}(T_u)$ **do**
    if $-\varepsilon < a+k-f$ and $b+k-f < \varepsilon$ **then**
      Append $[\min(a+k-f,0), \max(0,b+k-f)]$ to S_k
Merge S_{-1}, S_0, S_1 and S_2 into lexicographically ordered list S
return FILTER(S)

ascending order by lower bound *and* by upper bound (so that none of these intervals contains another). This invariant is maintained because we only push $[a,b]$ onto the stack when the stack is empty, or when the top $[a',b']$ of the stack satisfies both $a' < a$ and $b' < b$. Second, we only discard root error ranges that are not locally optimal. To see this, observe that an element $[a',b']$ is removed from the stack only when we scan an element $[a,b]$ with $a' \leq a$ (because $[a,b]$ follows $[a',b']$ in the scanning order) and $b \leq b'$ (otherwise we would stop popping). This implies $[a,b] \in [a',b']$. If $[a,b] = [a',b']$, we will now pop $[a',b']$ but push $[a,b]$ onto the stack again and the stack does not actually change. Otherwise, $[a,b]$ is contained in and smaller than $[a',b']$, so $[a',b']$ does not correspond to a locally optimal rounding and is rightfully removed from the stack. We only refrain from pushing $[a,b]$ when the top of the stack satisfies $a = a'$ and $b' < b$, so that $[a',b']$ is contained in and smaller than $[a,b]$ and therefore, the root error range $[a,b]$ is not locally optimal. □

Given two trees T'_1 and T'_2 that have the same root vertex r, but are otherwise disjoint as in Figure 4.5, we denote by $T'_1 \cup T'_2$ the union of the two trees; $T'_1 \cup T'_2$ also has root r.

Lemma 18 (merging error range sets). *Given $\mathscr{E}(T'_1)$ and $\mathscr{E}(T'_2)$ for two trees T'_1 and T'_2 whose root r is the only vertex that they have in common, we can compute $\mathscr{E}(T'_1 \cup T'_2)$ in $O(|T'_1| + |T'_2|)$ time.*

Proof. Consider a rounding $\widetilde{\omega}$ of $T'_1 \cup T'_2$ that consists of an ε-rounding of T'_1 with root error range $[a_1,b_1]$ and an ε-rounding of T'_2 with root error range $[a_2,b_2]$. For $i \in \{1,2\}$, let u_i and v_i be vertices in T'_i that determine the lower and upper bounds of the root error range $[a_i,b_i]$, that is: $a_i = e(r,u_i)$ and $b_i = e(r,v_i)$. The path composed of $\pi(u_1,r)$ and $\pi(r,u_2)$ is a path in $T'_1 \cup T'_2$ with $e(u_1,u_2) = a_1 + a_2$; similarly, we have $e(v_1,v_2) = b_1 + b_2$. It follows that $\widetilde{\omega}$ can be an ε-rounding only if $a_1 + a_2 > -\varepsilon$ and $b_1 + b_2 < \varepsilon$. These conditions are also sufficient, since any other path from a vertex $w_1 \in T'_1$ to a vertex $w_2 \in T'_2$ consists of a path with error $e(r,w_1) \in [a_1,b_1]$ and a path with error $e(r,w_2) \in [a_2,b_2]$, so the total error is within $[a_1+a_2, b_1+b_2]$.

Since T'_1, T'_2 and $T'_1 \cup T'_2$ have the same root, the root error range $E(T'_1 \cup T'_2, \widetilde{\omega})$ is the union of the root error ranges of T'_1 and T'_2, that is, $E(T'_1 \cup T'_2, \widetilde{\omega}) = [\min(a_1,a_2), \max(b_1,b_2)]$. We say $\widetilde{\omega}$ is of type 1 if $a_1 < a_2$, and of type 2 if $a_2 \leq a_1$.

We will now explain how to find a linear-size set S_1 of root error ranges for $T'_1 \cup T'_2$ that includes the root error ranges of all locally optimal roundings of type 1. Recall that such a root error range for $T'_1 \cup T'_2$ must stem from a root error range $[a_1,b_1] \in \mathscr{E}(T'_1)$ and a root error range $[a_2,b_2] \in \mathscr{E}(T'_2)$ that satisfy the following conditions: (i) $a_2 > a_1$ (condition for type 1); (ii) $a_2 > -\varepsilon - a_1$; (iii) $b_2 < \varepsilon - b_1$. The idea of the algorithm is to scan the root error ranges $[a_1,b_1] \in \mathscr{E}(T'_1)$ in ascending order while

maintaining pointers to: the first range $[a'_2,b'_2] \in \mathscr{E}(T'_2)$ that satisfies condition (i); the first range $[a''_2,b''_2] \in \mathscr{E}(T'_2)$ that satisfies condition (ii); and the last range $[a'''_2,b'''_2] \in \mathscr{E}(T'_2)$ that satisfies condition (iii)[3]. Note that as a_1 and b_1 increase, the first pointer ascends in $\mathscr{E}(T'_2)$, whereas the second and the third pointer descend in $\mathscr{E}(T'_2)$. Therefore we can scan all of $\mathscr{E}(T'_1)$ while maintaining the three pointers into $\mathscr{E}(T'_2)$ in linear time.

For any error range $[a_1,b_1]$ scanned from $\mathscr{E}(T'_1)$, consider now the range $[a_2,b_2] \in \mathscr{E}(T'_2)$ with $a_2 = \max(a'_2,a''_2)$, that is, the range pointed to by the furthest of the first two pointers. If $b_2 \leq b'''_2$ (that is, we are not past the third pointer), we include $[\min(a_1,a_2),\max(b_1,b_2)] = [a_1,\max(b_1,b_2)]$ in S_1. Note that we do not need to consider a combination of $[a_1,b_1]$ with any other range $[a,b] \in \mathscr{E}(T'_2)$ between $[a_2,b_2]$ and $[a'''_2,b'''_2]$: the resulting error range for $T'_1 \cup T'_2$ would be $[a_1,\max(b_1,b)]$ and include $[a_1,\max(b_1,b_2)]$, so it would be either a duplicate of $[a_1,\max(b_1,b_2)]$, or it would not be locally optimal.

In a similar fashion, we can find a linear-size set S_2 of root error ranges for $T'_1 \cup T'_2$ that includes the root error ranges of all locally optimal roundings of type 2. Finally we can merge S_1 and S_2 and filter out error ranges that are not locally optimal with the algorithm described in the proof of Lemma 17. The full algorithm is given by Procedure MERGEERRORRANGESETS. □

Algorithm 7: MERGEERRORRANGESETS($\mathscr{E}(T'_1),\mathscr{E}(T'_1)$)

Input: $\mathscr{E}(T'_1)$ and $\mathscr{E}(T'_2)$, each in ascending order, where T'_1 and T'_2 have a common root.
Output: $\mathscr{E}(T'_1 \cup T'_2)$ in ascending order.

```
Add sentinel [−ε,−ε] at the beginning of E(T'_1) and E(T'_2)
Add sentinel [ε,ε] at the end of E(T'_1) and E(T'_2)
S_1 ← empty list
Let p_1 point to the first element of E(T'_2)
Let p_2 and p_3 point to the last element of E(T'_2)
foreach [a_1,b_1] in E(T'_1) except the sentinels do
    while element [a_2,b_2] pointed at by p_1 violates a_2 > a_1 do
        p_1 ← pointer to successor
    while predecessor [a_2,b_2] of element pointed at by p_2 satisfies a_2 > −ε − a_1 do
        p_2 ← pointer to predecessor
    while element [a_2,b_2] pointed at by p_3 violates b_2 < ε − b_1 do
        p_3 ← pointer to predecessor
    Let [a'_2,b'_2],[a''_2,b''_2],[a'''_2,b'''_2] be the elements pointed at by p_1,p_2,p_3
    if max(b'_2,b''_2) ≤ b'''_2 then Append [a_1,max(b_1,b'_2,b''_2)] to S_1
Compute S_2 in a similar manner
Merge S_1 and S_2 into a lexicographically ordered list S
return FILTER(S)
```

To decide whether a tree T admits an ε-rounding, we compute $\mathscr{E}(T_u)$ for all subtrees T_u of T bottom-up.

Specifically, if u is a leaf, $\mathscr{E}(T_u) = [0,0]$. If u is an internal vertex with a single child v, then $T_u = T'_v$ and we compute $\mathscr{E}(T_u) = \mathscr{E}(T'_v)$ from $\mathscr{E}(T_v)$ with the algorithm of Lemma 17. If u is an internal vertex with two children v and w, we first compute $\mathscr{E}(T'_v)$ and $\mathscr{E}(T'_w)$ from $\mathscr{E}(T_v)$ and $\mathscr{E}(T_w)$, respectively, with the algorithm of Lemma 17, and then we compute $\mathscr{E}(T_u) = \mathscr{E}(T'_v \cup T'_w)$ with the

[3] Consider $\mathscr{E}(T'_2)$ augmented with sentinel ranges $[-\varepsilon,-\varepsilon]$ and $[\varepsilon,\varepsilon]$.

algorithm of Lemma 18. Finally, if u is an internal vertex with more than two children, we first compute $\mathscr{E}(T'_v)$ from $\mathscr{E}(T_v)$ for each child v. Then we organize all children in a balanced binary merge tree M with the children of u at the leaves ; for a vertex x in M, let $C(x)$ be the children of u in the subtree of M rooted at x. With vertex x we associate the error range set $\mathscr{E}(\bigcup_{v\in C(x)} T'_v)$. We process the merge tree M bottom-up, using the algorithm of Lemma 18 for each internal vertex x of M to compute $\mathscr{E}(\bigcup_{v\in C(x)} T'_v)$ from the error range sets associated with the children of x. The error range set computed for the root of M constitutes $\mathscr{E}(T_u)$.

Ultimately, we compute $\mathscr{E}(T_r)$. If and only if this error range set is non-empty, T admits an ε-rounding.

We say the *effective* height of the tree T is the height it would have when all internal vertices with more than two children were replaced by their binary merge trees. The algorithms of Lemmas 17 and 18 take time linear in the size of the subtrees that are being processed. Thus, if T has n vertices and effective height h, the above algorithm to compute $\mathscr{E}(T_r)$ runs in $\mathrm{O}(nh)$ time. This proves:

Theorem 10. *Given an edge-weighted tree T of n vertices and an error tolerance ε, one can decide in $O(n^2)$ time whether T admits an ε-rounding.*

To find the minimal maximum rounding error, we first compute the lengths of all $\mathrm{O}(n^2)$ simple paths in the tree in $\mathrm{O}(n^2)$ time. We can do so with a bottom-up algorithm that computes for each vertex u the lengths of all paths in T_u, and passes on the lengths of all paths in T_u that end in u to the parent of u. Each path produces up to four candidate values for the maximum rounding error, namely, for $k \in \{-1,0,1,2\}$, the absolute value of (k minus the fractional part of the path length). We sort all of these candidate values in $\mathrm{O}(n^2 \log n)$ time. Finally we find the smallest error tolerance for which the decision algorithm says yes by binary search, using $\mathrm{O}(\log n)$ calls to the decision algorithm, which takes $\mathrm{O}(n^2 \log n)$ time in total.

Corollary 9. *Given an edge-weighted tree T of n vertices, we can compute a rounding of T that minimizes the maximum absolute rounding error on any simple path in the tree in $O(n^2 \log n)$ time.*

4.4 Conclusion and Outlook

In this chapter, we examined a rounding problem on edge-weighted graphs where three constraints had to be met. On the one hand, the absolute change in weight along each shortest path had to be less than a given error threshold ε. On the other hand, a shortest path had to remain shortest after rounding the edge-weights. We further imposed the restriction that a shortest path, after the rounding, had to have already been shortest beforehand. We discussed the question how the difficulty of this problem depends on the value of ε: By reduction from 3-SAT, we proved that the problem is NP-hard for $\varepsilon \in (3/4; 1]$, even in its simplest form. However, if the graph is a tree, the problem can be solved in polynomial time. The conditions of our NP-hardness construction raise several questions.

First, note that we motivated the study of the ε-rounding problem with applications of graphs that are, typically, almost planar. These graphs are usually not as simple as trees, but neither they are like the graph in our NP-hardness proof where most pairs of vertices are connected by a direct edge. To deal with realistic graphs in data structures for shortest-path queries, Storandt [96] proposes to augment graphs with additional edges. The edges can represent paths of many edges in the original graphs without suffering from accumulated rounding errors; however, for the map drawing application, which we mentioned in the introduction, such an approach would not be suitable. In this application we have to keep rounding the edge-weights of the original graph. Where then, between trees and

almost-complete graphs, lies the boundary between easy and hard? So far, we have been unable to reduce 3-SAT, even planar 3-SAT, to a planar instance of an ε-rounding problem. It might be that in a (near-)planar graph, the dependencies between shortest paths between different pairs of vertices are so strong that the problem becomes easy to decide, as it is the case with trees. Possibly, a first step in this direction would be to develop an efficient algorithm for graphs that consists of a single cycle or a tree with one additional edge. To improve our understanding of the structure of the problem, we may also try to get a subquadratic algorithm for trees or to develop parametrized algorithms for general graphs, e. g., where the parameter limits the number of paths along which the conditions have to hold.

Second, we may further examine how NP-hardness of the problem depends on the choice of ε. In our NP-hardness proof we assumed $\varepsilon \in (3/4; 1]$. Note that for the case where $\varepsilon < 1/2$ the problem is trivial for arbitrary graphs: Each edge can only be rounded in one way (if at all); then, one can verify in polynomial time whether necessary conditions are satisfied. However, neither for $1/2 \leq \varepsilon \leq 3/4$ nor for $\varepsilon > 1$ we know how to construct a working variable gadget. For which of these values does the problem remain NP-hard? For which does it become easy to solve?

Third, we might reconsider the exact conditions of an ε-rounding. Observe that our results hold even for the rounding problem in which we only ensure Condition 1 (the absolute change in weight) and ignore Condition 2 and 3 (shortest paths remain shortest paths): the graphs in our NP-hardness construction are such that the second and third conditions are implied by the first. Thus, by itself, dropping the requirement that shortest paths should be preserved does not make our problem easier. Funke and Storandt [40] consider the rounding problem with different conditions. It is not clear whether this reduces the theoretical complexity of the problem as compared to the conditions of our approach. Funke and Storandt observe that many rounding problems are NP-hard, but do not prove this specifically for shortest-path-preserving rounding problems. They describe an integer linear program that they find to be too expensive to solve even for small graphs, and then describe and evaluate a greedy rounding heuristic.

Although it is interesting to aim for exact solutions of the problem, it might be more relevant for many applications to go for approximate solutions instead. A crucial step is therefore to examine whether the problem remains NP-hard for values of ε (slightly) larger than one. On trees, we can compute a 2-rounding in linear time; do similar approximations also exist for other graph classes? This would be of particular interest since some graph algorithms are significantly faster when applied to integer-weighted instead of real-weighted graphs [47, 98, 99, 102]. For example, consider classical problems as the single-source shortest path (SSSP) or the all-pairs shortest path (APSP) problem. For the SSSP problem, Thorup [99] showed that it can be solved for directed graphs in deterministic $O(E + V \log\log V)$ time if the weights of the edges are non-negative integer values. Note that Thorup's runtime bound only depends on the number of vertices and not necessarily on the largest weight. For the APSP problem, the currently best runtime bound is $O(EV + V^2 \log\log V)$ for integer-weighted, directed graphs [47]. Thus, strong ε-roundings could lead to runtime improvements for classical problems on real-weighted graphs, even if ε is large: Compute an ε-rounding efficiently in a preprocessing step and apply the algorithms for integer-weighted graphs afterwards.

We conclude with a note on the computational model used in this chapter. The polynomial time reduction presented in Section 4.2 holds for the standard RAM model: all weights of the resulting graph are rational and, to solve the original 3-SAT problem, we may choose a rational value $\varepsilon \in (3/4; 1]$. With respect to the algorithm on trees, the runtime analysis presumes a RAM model which supports the floor operation for rational-valued input. If, however, the input is real-valued, we require a stronger model already to store the input; a real RAM, e. g., would be sufficient.

Chapter 5

A Universal Fire-Expansion Model

This chapter examines the search for a *single*, specific configuration in a new, two-dimensional, *discrete* fire-expansion model: Given two cells s, t where cell s is initially set on fire and the fire expands in discrete time steps, will the *single* target cell t eventually ignite?

The fire-expansion model provides a simple and more realistic framework for the study of fire-expansion or fire-fighting problems in large varied terrains. The new model builds on existing ones by introducing two integer parameters to each cell, which model the cell's resistance against ignition and its remaining fuel. This simple extension adds a lot of computational power to this model: We prove by reduction from two-register machines that the fire-expansion model is Turing-universal, i. e., it is capable of simulating the computations of every Turing machine. Thus, the halting problem for Turing machines implies that the initial question is undecidable on Turing machines, i. e., no general algorithm exists that can solve every instance of this decision problem within any finite number of steps. Since the new fire-expansion model is related to concepts like cellular automata and artificial neural networks, both of these concepts are briefly reviewed and their relation to the model is discussed.

5.1 Formal Problem Statement and Related Work

Predicting the spread of fire in practice is a highly complex task that involves many parameters one can neither foresee nor control. For the formal study of such problems, several models have been suggested and investigated in different communities.

In Theoretical Computer Science and Mathematics, models have been examined where fire spreads in a polygon [69], in the Euclidean plane [17, 68, 63], or along edges of a graph [27, 39]. Usually, the challenge is to limit or stop the fire's expansion, e. g., by building barriers or placing fire-fighters. Research in these models normally focuses on proving tight lower and upper bounds on what can be achieved with a limited resource: In continuous models, researchers have been analysing the building speed of barriers which slow down or even stop the fire's expansion; in discrete models, the number of fire-fighters available to block/contain/extinguish the fire has been considered. Tight bounds are only available for simple cases in these models, e. g., see [63] or [34] for a survey.

In other communities, models have been developed to predict a fire's expansion in a given terrain. To forecast as realistic as possible, some models incorporate thermodynamic or chemical parameters as well as weather conditions, e. g., wind speed and direction. Some of the models are capable of distinguishing between fires at different heights such as ground fires and crown fires. For a survey on theoretical and (semi-) empirical models, see [85].

(i) Initial values (ii) $t = 0$ (iii) $t = 1$ (iv) $t = 2$ (v) $t = 3$

Figure 5.1: Fire-expansion in a section of 3×3 cells. In each cell, the corresponding x- and y-value is given in the form $x;y$ at the beginning of the designated time step t. The cell's state is indicated by colour: Red for burning, white for alive, and grey for dead. At the transition to the next time step, the propagation rules are applied which affects the x- and y-values: (i) The initial x- and y-value pairs; (ii) a single cell is set on fire; (iii) the fire spreads to neighbouring cells; (v) the fire-expansion stopped.

We introduce the following new model with the aim to provide a simple, theoretical framework for the study of fire-expansion and fire-fighting problems in large, varied terrains.

Definition 12 (fire-expansion model). *A* layer $\mathscr{L}$ *is a regular tiling of the plane into square cells. The state of cell c at time $t \in \mathbb{N}_0$ is given by two non-negative integers, $x(c,t)$ and $y(c,t)$. Cell c is called* burning *at time t if $x(c,t) = 0$ and $y(c,t) > 0$ hold;* alive *if $x(c,t) > 0$ and $y(c,t) > 0$; or* dead *if $y(c,t) = 0$ holds. At the transition from time t to $t+1$, the state of cell c changes as follows:*

- *If c is alive at time t, then $x(c,t+1) \coloneqq \max\{x(c,t) - b, 0\}$, where b denotes the number of direct neighbours of c burning at time t;*
- *if c is burning at time t, then $y(c,t+1) \coloneqq y(c,t) - 1$.*

An instance *of the* fire-expansion model *is a finite description of the regular cell-graph which consists of a fixed number of layers. Each cell of a layer contains an individual pair of integer weights. The layers are stacked on top of each other whereby each cell of the bottom layer is adjacent to the cell directly above and vice versa.*

Intuitively, x and y describe the (diminished) resistance against ignition and the (remaining) fuel of an individual cell at time t. Choosing suitable values for the cells, one can model natural properties of a given terrain: different types of ground and fuel; natural obstacles such as mountains or rivers. Thus, a cell of dry grassland might get small integers for both values such that it catches fire easily and burns down quickly. In contrast, we expect both values to be comparatively high for a moist forest such that the forest keeps burning for quite a while, once it caught fire. Figure 5.1 exemplifies the expansion of a fire over time in a bounded section of a single layer.

The fire-expansion model extends previously suggested discrete models in two main aspects. First, instead of a single layer of cells in the Euclidean plane, we allow for multiple layers which are stacked on top of each other and linked horizontally. Second, each cell obtains two individual integer values which model the resistance against ignition and the fuel of the cell. These simple extensions provide the flexibility to simulate the fire-expansion in existing models where fire spreads, e. g., in grids [27]. These simple extensions also add a lot of computational power as we prove in this chapter: With only two layers of cells, the model is capable to simulate every Turing machine, although, cells are unable to restore their initial state. In contrast to the results of this chapter, variants of the new model have been considered where only a *finite* number of cells in a *single* layer is considered. Some interesting algorithmic fire-fighting questions in this variant are discussed in [66, 67]; moreover, these problems are examined in the doctoral thesis of Barbara Schwarzwald [90].

The individual pairs of integer values stored in the cells immediately imply the following major differences to existing models. First, not all cells necessarily catch fire when adjacent to a burning cell, which implies that natural barriers can exist hidden in the cells values. Second, an expanding fire can die down on its own without the need to block or extinguish burning cells at all. Last but not least, different cells emit different amount of energy once burning; hence, in a dynamic version where a limited number of cells can be blocked per step to fight the fire's expansion, one has to carefully choose which of the cells to block.

Organisation of this chapter. In Section 5.2, we discuss how the model is related to other concepts like cellular automata and artificial neural networks. In Section 5.3, we prove that the new fire-expansion model can be used to simulate the computations of every Turing machine. We conclude with Section 5.4.

5.2 Related Work

The fire-expansion model is closely related to two powerful concepts which have been extensively studied in the past: *Cellular Automata* (CAs) and *artificial Neural Network* (NNs). This section introduces both concepts and explains how they can be used to simulate the fire-expansion model.

Although the fire-expansion model can be simulated via CAs or NNs, it is still reasonable to examine the new model. First, it links up two fields that might not have seem closely related before: Theoretical fire-fighting and Computability theory. This might explain why many computational questions in this simple model are surprisingly difficult to solve: With only a single layer and a limited number of cells, questions are always decidable but seem to be efficiently solvable only for special cases [67, 90]; with only two layers of cells, the fire-expansion is capable of simulating every Turing machine, which implies that many questions are undecidable. Moreover, the proof of this simulation result requires no complicated tools; it uses simple gadgets which are assembled to describe the desired instance of the fire-expansion model. Nevertheless, the relation between the fire-expansion model and CAs, NNs gives rise to several questions, which we pick up in Section 5.4.

5.2.1 On the Relation to Cellular Automata

In the following, we briefly introduce and review the concept of cellular automata; the notation is similar to [43, 45]. A cellular automaton is a deterministic, dynamical system that consists of an array of identical finite state machines which change their state simultaneously at discrete time steps. It can be described as a tuple $(G, Q, \mathcal{N}, \delta)$ where: $G = (V, E)$ is a regular graph, $\mathcal{N}$ defines a (reflexive) neighbourhood relation on V, Q is a finite set of states, and δ is a transition function. Often, the vertices of V are also called *cells* and the transition function is called the *update rule*. An identical finite-state machine M, with state set Q and transition function δ, is placed at every $v \in V$. With k we denote the size of Q and with n the size of the neighbourhood for v, i. e., $k \coloneqq |Q|$ and $n \coloneqq |\mathcal{N}_v|$. Finally, the transition of M from time t to $t+1$ is defined by the mapping $\delta : Q \times (q_1, \dots, q_n) \to Q$, which depends on M's state and M's neighbours' states $(q_1, \dots, q_n) \in Q^n$ at time t. Note that, at every time step, the transition function is applied to all cells simultaneously.

Subsequently, we restrict our attention to the following popular parameters for the definition of a cellular automaton. We consider cellular automata for which G is the d-dimensional integer lattice $\mathbb{Z}^d$ and $d \in \{1, 2\}$. The neighbourhood relation is defined via the L_∞ or the L_1 metric: The *Moore*

neighbourhood of radius r for a vertex $v = (v_1, v_2) \in \mathbb{Z}^2$ is defined as the set

$$\mathcal{N}_v := \left\{ z \in \mathbb{Z}^2 \;\middle|\; |v_1 - z_1| \leq r \;\wedge\; |v_2 - z_2| \leq r \right\};$$

the *von Neumann neighbourhood* of radius r for a vertex at $v = (v_1, v_2) \in \mathbb{Z}^2$ is defined as the set

$$\mathcal{N}_v := \left\{ z \in \mathbb{Z}^2 \;\middle|\; |v_1 - z_1| + |v_2 - z_2| \leq r \right\}.$$

Many popular CAs have only two states ($k = 2$), which suffices to model binary extremes such as 0/1, dead/living, or black/white.

Cellular automata have been extensively studied in the past. In [105], Wolfram presents an impressive study on CAs and how their update rules behave on various input patterns. Wolfram suggests a classification of CAs according to their asymptotic behaviour on finite input patterns; however, this classification scheme has crucial weaknesses as observed by several authors [9, 10, 16, 25] who propose different classification schemes. Common to most of the classification schemes seems to be that CAs which are capable of Turing-universal computation form the most restricted class; thus, a CA that simulates the fire-expansion model belongs to this class of CAs. The fact that (even one-dimensional) CAs can simulate arbitrary Turing machines is well known, e. g., see [10, 55] or [43, p. 70]; however, these proofs usually hinge on the fact that cells can restore their initial states. Note that this is not possible for cells in our fire-expansion model.

The definition of the new fire-expansion model bears strong resemblance to cellular automata. This impression comes from three common features: First of all, both definitions rely on regular graphs which specify where computational units are located. Second, in both models, the transition function δ is identical for all computational units. Finally, the transition function is local for both models. In fact, every instance of the fire-expansion model can be simulated by a CA with $d = 3$ and the $r = 1$ von-Neumann neighbourhood. However, following this approach implies that the number of states of the CA depends on the x- and y-values stored in the cells of the fire-expansion model. Thus, the resulting CA requires large components (set of states, transition function) in contrast to the simple and intuitive instance of the fire-expansion model.

In the following, we briefly introduce three popular cellular automata and discuss their relation to the fire-expansion model.

The first (and probably most popular) cellular automaton is *Game of Life* (GoL) by John Conway [42]. The parameters of GoL are $d = 2$, $k = 2$, and the $r = 1$ Moore neighbourhood; the transition function is defined by the following rules: A dead cell at time t becomes alive at time $t + 1$ if exactly 3 of its 8 neighbours are alive; a living cell survives if exactly 2 or 3 neighbours are alive at time t and dies from overpopulation otherwise. Initially, at time $t = 0$, all cells are dead except for a finite population. Note that the transition is quite different to the update rules of the fire-expansion model: Cells can die from overpopulation in GoL for which there is no equivalent transition in the fire-expansion model; moreover, dead cells can be reborn which is impossible in the fire-expansion model. To prove that GoL can simulate every Turing machine, one crucially depends on the fact that cells can restore their initial state.

Another popular cellular automaton is often referred to as Rule 110. The choice of it's parameters are $d = 1, k = 2$, and $r = 1$, which is particularly interesting since it minimizes all three parameters of a CA. The evolution of an initial population of such a CA crucially depends on the update rule. For a systematic study of the update rules, Wolfram [104] suggested the following scheme to enumerate all of them: Due to the choice of the neighbourhood, the input of δ for a cell $z \in \mathbb{Z}$ are the states of the cells $z - 1$, z, $z + 1$; thus, each input is one of eight binary strings $\{0, 1\}^3$. All input strings can be

sorted in lexicographical ascending order to obtain $s_1 \leq \cdots \leq s_8$; concatenating the binary images $\delta(s_1)\ldots\delta(s_8)$ in the same order, uniquely defines a binary number of eight bits. This binary number corresponds to a decimal number which ranges from 0 to 255 and defines the *number of the update rule*. Some of the update rules are not difficult to understand and the evolution of an initial population can be predicted easily. However, this is not the case for Rule 30 and Rule 110: Rule 30 seems to produce unpredictable patterns starting from simple initial states [45, 105]; Rule 110 equips the CA with unlimited computational power, i. e., based on Rule 110, the CA is able to simulate every Turing machine as Cook proved in [22]. Note that this is only possible since cells can restore their initial state, which is impossible in the fire-expansion model. Nevertheless, as Baldwin already states in [9], Cook's result is extraordinary: It shows that all three parameters of CA complexity (i. e., dimension, number of states and radius) can be minimized without loosing the CA's power to simulate every Turing machine.

Finally, a popular cellular automaton, which seems closely related to our model, is *Wireworld*, as Dewdney calls it in [28]. The parameters of this CA are $d = 2, k = 4$, and the $r = 1$ Moore neighbourhood. The four states of the cells in the plane model are called: background, wire, electron-tail, and electron-head. The names of the states become immediately already clear when looking at the definition of the update rules: Electron-head cells turn electron-tail; electron-tail cell turn wire cell; wire cells turn electron-head cell if exactly one neighbour is electron-head; finally, background cells never change their state. Thus, this CA allows to simulate an arbitrary Turing machine: On the two-dimensional array of background cells, one can draw necessary wires, logical gates, and memory registers. Background cells never change their state and serve to isolate components from each other. Note that also this construction crucially depends on the fact that wire cells can restore their initial state. This, however, is impossible in our model.

5.2.2 On the Relation to Automata Networks and Artificial Neural Networks

Important models which build on cellular automata are *automata networks* and discrete, *artificial neural networks*. In the following, both models are briefly described; for a formal definition see [43].

Automata networks extend CAs in two important aspects: First, the underlying graph G does no longer need to be regular; it suffices that G is countably infinite, locally-finite, and directed. Second, the finite-state machines located at vertices of G do no longer have to be identical; instead, there can be a finite number of different machines, enumerated by $M_1, \ldots, M_m$, which makes it necessary to define a specific transition function δ_i for each M_i. In a nutshell, an automata network does not have to be as homogeneous as a CA; yet, the network allows for a finite description and transitions are performed locally.

Artificial neural networks (NNs) are a special case of automata networks with the following properties. The underlying, directed graph $G = (V, E, \omega)$ is edge-weighted and the set of states has an additive-multiplicative structure. Moreover, for every vertex $v \in V$, the state of v at the next time step depends only on $\sum_{(u,v)\in E} \omega(u,v) \cdot q_u(t)$ where $q_u(t)$ denotes the state of vertex u at time t. That is, the next state of v depends on the weighted sum of the current state of all cells which have v as a neighbour. Thus, the transition function is oblivious to where exactly input comes from.

Due to outstanding achievements in practice, NNs have experienced several renaissances since the early attempts by McCulloch and Pitts [78], in 1943. In the late 1980s and early 1990s, they have been successfully applied to solve practical problems such as to recognise speech or patterns, to simulate artificial life, or to solve combinatorial problems, e. g., see [43, p. 113 f.] or [82, pp. 62 ff.]. Again, in the late 2010s, they attracted a lot of attention when they have been successfully applied to

problems like the board game Go. For many years, this game has been considered as a domain where humans outplay computers.

From a theoretical point of view, there are three important results to mention: First of all, the class of Turing-computable functions (and equivalent classes) is contained in the class of functions that can be computed via NNs: Turing machines can be simulated by CAs, which are special cases of NNs. Moreover, this inclusion is proper since, e. g., the halting problem for Turing-machines[1] is solvable by means of NNs, cf. [43, p. 106 f.]. Note that this does *not* invalidate the Church-Turing thesis; the underlying graph of a NN can be infinite whereas all defining elements of Turing machines are finite. Second, similar to a universal Turing machine, there exists a universal NN which is capable of simulating every given NN on every given starting configuration. Finally, one can prove the existence of problems that are unsolvable to NNs; the proof employs a *diagonal argument*. Finally, all three models, NNs, automata networks and CAs are computationally equivalent when restricted to graphs of finite bandwidth; cf. [43].

The fire-expansion model as defined in Section 5.1 bears strong resemblance to artificial neural networks. In addition to the similarities with CAs, this impression comes from two features: First, the cells of the fire-expansion model can be considered as different computational units placed at vertices of a directed graph. Second, in each time step, the local transition function in the fire-expansion model depends only on the number of burning neighbouring cells.[2]

As described above, an instance of the fire-expansion model can be simulated by a CA; since NNs extend CAs, it can also be simulated by NNs. Such a NN can be constructed as follows: A directed graph for the NN can be obtained by creating a vertex for every cell in the cell-graph; to obtain the set of edges, create two directed edges (u,v) and (v,u) with unit weight for every pair of vertices u,v if the corresponding cells c_u, c_v are adjacent in the cell-graph. However, following this approach implies that the number of states of the NN depends on the x- and y-values stored in the cells of the fire-expansion model. Thus, the resulting NN requires large components (set of states, transition function) in contrast to the simple and intuitive instance of the fire-expansion model.

5.3 Simulation of Turing Machines with the Fire-Expansion Model

By reduction from two-register machines, we prove that, with only two layers, the fire-expansion model can simulate every Turing machine M. For the proof we proceed in two steps: In a first step, we introduce the notion of two-register machines in Section 5.3.1. We further argue that, for every Turing machine M, there exists a two-register machine *2M* which computes the same function. Moreover, we describe a nice, geometric interpretation of two-register machines in Section 5.3.2. In a second step, we use the idea of this geometric interpretation to construct an instance $\square_{2M}$ of the fire-expansion problem for every given two-register machine *2M*. The instance $\square_{2M}$ has an important property: Computations of *2M* correspond to fire-expansions in the cell-graph of $\square_{2M}$ and vice versa.

5.3.1 A Note on Two-Register Machines

We define two-register machines in the style of Boerger et al. [15, p. 28 f.]. A two-register machine *2M* is a triple $(r_1, r_2, \mathscr{P})$ where: r_1, r_2 are two registers that contain an arbitrarily large, non-negative integer-value each and $\mathscr{P}$ is a finite *program*, i. e., a finite sequence of instructions $(I_1, \ldots, I_n)$. Each

[1] In general, this holds for every recursively enumerable language.

[2] For a single transition it is irrelevant which of the neighbouring cells are burning; although, this can make a difference over time due to different x- and y-values in the cells.

instruction can be uniquely identified via its index. We assume that I_n is a special instruction that causes *2M* to halt; all other instructions are either addition instructions or subtraction instructions. An *addition instruction* is a triple (i,r,j) where i is the index of the instruction, $r \in \{r_1,r_2\}$ specifies one of the registers, and j is the index of the next instruction. When processing this instruction, *2M* increments the value of r by one and proceeds with I_j. A *subtraction instruction* is a quadruple (i,r,j,k) where i is the index of the instruction, $r \in \{r_1,r_2\}$ specifies one of the registers, and j,k are indices of (possibly) following instructions. When processing this instruction, *2M* checks first whether the content of r is zero: If this is the case, then *2M* proceeds with I_j; otherwise, *2M* decrements r by one and proceeds with I_k. Note that, using subtraction instructions, it is possible to branch the control flow and simulate conditional statements. A *configuration C* of a two-register machine *2M* is a triple (b_1,b_2,I_i) where $b_1,b_2 \in \mathbb{N}$ are the contents of the registers r_1,r_2 and I_i is the instruction that *2M* is going to process next.

Minsky [79] proved that a two-register machine program *2M* exists for every Turing machine M which computes the same function. The idea of the proof is as follows.[3] W. l. o. g. we may assume that M has a single, semi-infinite tape. Since the tape's content σ is finite at any time, it can be represented as a single, non-negative integer which we denote by $\mathrm{val}(\sigma)$. Moreover, the position of the tape's head can be uniquely defined via the index of the character of σ to which the head currently points to; i. e., if the head points to σ_i, this value is equal to i. Thus, a configuration of M can be uniquely encoded with the integer $2^{\mathrm{val}(\sigma)}3^{2^i}$. The two-register machine *2M* stores this integer representation of M's configuration in the first register r_1 and uses r_2 to simulate a transition of M.

5.3.2 Geometric Interpretation of Two-Register Machines

Two-register machines have a nice geometric interpretation. Consider the boxes[4] generated by the $\mathbb{N}_0 \times \mathbb{N}_0$ cell graph and illustrated in Figure 5.2. A box b can be uniquely addressed by a tuple of integers coordinates (b_1,b_2), where $(0,0)$ corresponds to the bottom left most box; we also write $b = (b_1,b_2)$. The tuple of coordinates of each box can be interpreted as the registers of a two-register machine *2M*; i. e., the box with coordinates $(3,2)$ corresponds to *2M*'s register contents $(r_1,r_2) = (3,2)$.

Given a two-register machine *2M*, we construct a locally-finite, directed graph $G_{2M} = (V,E)$ as follows.[5] The infinite set of vertices V contains a vertex v_i^b for every box b and every instruction I_i; note that the number of vertices per box is finite, since the program of *2M* is finite. The infinite set of edges E consist of directed edges between vertices of the same or neighbouring boxes. For each box $b = (b_1,b_2)$, the edges are defined according to the following rules:

- For each addition instruction $I_i = (i,r_1,j)$, add a directed edge $(v_i^{(b_1,b_2)}, v_j^{(b_1+1,b_2)})$;
- for each addition instruction $I_i = (i,r_2,j)$, add a directed edge $(v_i^{(b_1,b_2)}, v_j^{(b_1,b_2+1)})$;
- for each subtraction instruction $I_i = (i,r_1,j,k)$, add a directed edge $(v_i^{(b_1,b_2)}, v_k^{(b_1-1,b_2)})$ if $b_1 > 0$ otherwise add $(v_i^{(b_1,b_2)}, v_j^{(b_1,b_2)})$;
- for each subtraction instruction $I_i = (i,r_2,j,k)$, add a directed edge $(v_i^{(b_1,b_2)}, v_k^{(b_1,b_2-1)})$ if $b_2 > 0$ otherwise add $(v_i^{(b_1,b_2)}, v_j^{(b_1,b_2)})$.

[3] In [79], Minsky uses two-tape non-writing Turing machines; these machines consist of two semi-infinite tapes which are completely empty and have a single head per tape. The content of a tape is encoded by the head's position on the tape.

[4] We use the term *box* to underline the difference to *cells* of the fire-expansion model which contain the x and y-values.

[5] It is well known that such a graph G_{2M} can be constructed within the $\mathbb{N}_0 \times \mathbb{N}_0$ cell graph. For example, Dieter Rödding presented this type of construction in lectures on Mathematical Logic at the University of Münster between 1980 and 1984.

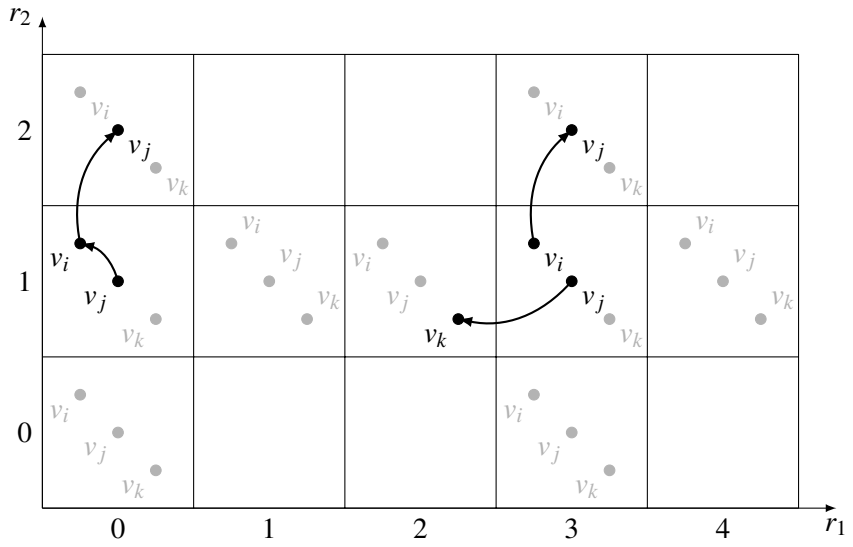


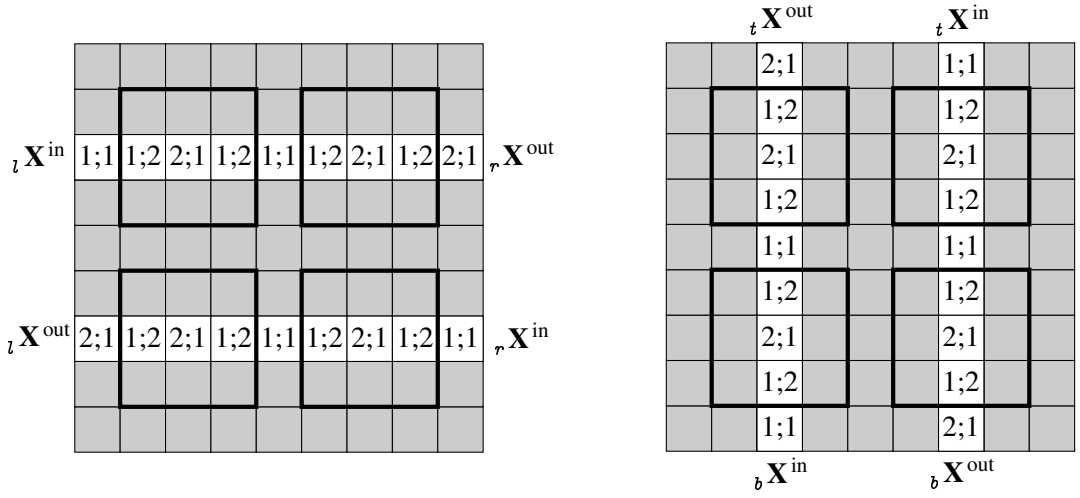
Figure 5.2: A sketch of how G_{2M} is constructed and embedded into the plane. Note that in order to improve the legibility: superscripts have been omitted; vertices are only given for the instructions I_i, I_j, I_k and only for the boxes with coordinates $(3,1)$, $(0,1)$ and their neighbours; edges were only drawn for the addition instruction $I_i = (i, r_2, j)$ and the subtraction instruction $I_j = (j, r_1, i, k)$ and only for vertices within the boxes with coordinates $(3,1)$ and $(0,1)$.

The resulting graph G_{2M} has several nice properties. First of all, although V and E contain infinitely many elements, the description of G_{2M} is finite. Moreover, the number of incoming and outgoing edges per box is finite, since the program of $2M$ is finite. Second, it can be intuitively drawn in the boxes as indicated in Figure 5.2. Finally, by construction of G_{2M}, every vertex has exactly one outgoing edge, except for the ones that correspond to $2M$'s halting instruction; this stems from the fact that computations of $2M$ are deterministic and implies that a directed path between two vertices of G_{2M} is unique. In fact, one can prove the following observation via induction over the length (number of edges) of the path.

Observation 5. *There is a directed path from $v_i^{(b_1,b_2)}$ to $v_j^{(b_1',b_2')}$ in G_{2M} if and only if the two-register machine $2M$ reaches configuration (b_1', b_2', I_j) when started in configuration (b_1, b_2, I_i), i. e., $2M$ is started with initial register contents $(r_1, r_2) = (b_1, b_2)$ and I_i as the first instruction.*

5.3.3 Simulating Two-Register Machines via Fire-Expansion

Subsequently, we reduce a given two-register machine to an instance of the fire-expansion model; the idea of the reduction is as follows. For a two-register machine $2M$, we use the idea of the geometric interpretation to systematically construct G_{2M} within the two layers $\mathscr{L}_1$, $\mathscr{L}_2$ of the cell-graph; we denote the resulting instance of the fire-expansion model with $\square_{2M}$. The idea of this construction is to map edges of G_{2M} to sequences of alive cells called *wires*. For simplicity, wires in $\mathscr{L}_1$ are sequences of horizontally aligned cells; in $\mathscr{L}_2$, wires are sequences of vertically aligned cells. To prevent a fire on one wire from leaping over to a parallel one, we use cells which are dead from the very beginning. Finally, $\square_{2M}$ has a property similar to Observation 5 for specific cells c, c' in $\square_{2M}$ and specific configurations C, C' of $2M$: Setting cell c on fire, the fire follows particular wires and



(i) Layout of the cells in $\mathscr{L}_1$.

(ii) Layout of the cells in $\mathscr{L}_2$.

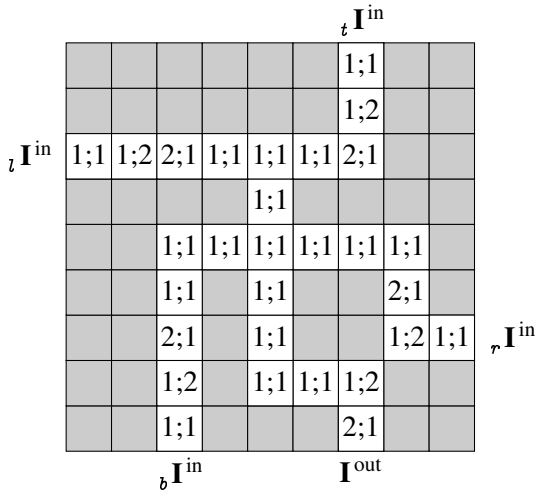
Figure 5.3: A crossing gadget with its initial x and y-values given in the the cells. The x- and y-values of grey shaded cells are both zero and have been omitted for the sake of legibility.

eventually ignites c' if and only if $2M$ reaches configuration C' within a finite number of steps when started in configuration C. Since $2M$ has a finite descriptions, so do G_{2M} and $\square_{2M}$.

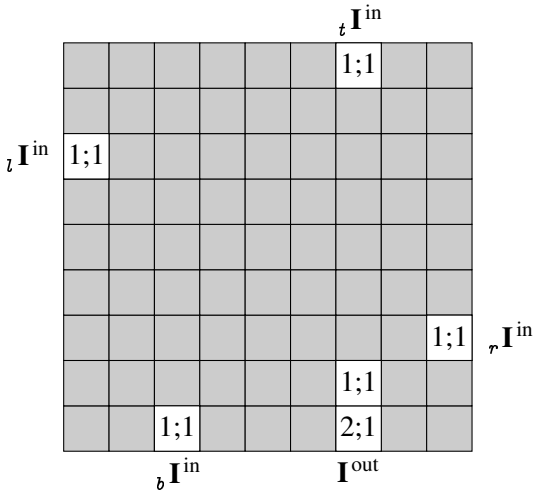
To describe the details of the reduction, we proceed as follows. The geometric interpretation as indicated in Figure 5.2 already provides a coarse subdivision of $\mathbb{N}_0 \times \mathbb{N}_0$ into boxes; we further subdivide the boxes into the cells of $\square_{2M}$. To specify the cells of each layer $\mathscr{L}_1$, $\mathscr{L}_2$ and their x-y value-pairs, we use the following types of gadgets: the *crossing gadget* **X**, the *instruction gadget* **I**, the *program gadget* **P**, the *connector gadget* **C**, and the *box gadget* **B**. A lemma describes the essential feature of each gadget. Proof sketches for the lemmata are given in Appendix C.

Throughout the description, we use the following notation. We say a cell c *ignites* at time t if and only if c's state is alive at time $(t-1)$ and burning at time t. Each gadget is a connected subset S of cells in $\mathscr{L}_1$, $\mathscr{L}_2$ which forms a rectangle in each layer and every cell in $\mathscr{L}_1$ is adjacent to exactly one cell in $\mathscr{L}_2$ (the cell directly above or below). A gadget's cell $c \in S$ is called a *boundary-cell* if it is adjacent to a cell $c' \notin S$; all other cells of S are *interior* cells. Since S forms a rectangle, we use the subscripts $_l$, $_r$, $_t$, and $_b$ to denote the boundary-cells at the left, right, top, and bottom edge of S. Moreover, a boundary-cell with weights $(1;1)$ is called an *input-cell*; similarly, a boundary-cell with weights $(2;1)$ is called an *output-cell*. To improve readability in the sketches of gadgets, grey shaded cells indicate cells with weights $(0;0)$, i. e., cells in the state dead. A *wire* is a path in the cell-graph where each of its cells is alive and can only ignite its direct predecessor or direct successor. If a wire is connected to an input-cell of a gadget, it is called an *input-wire* of the gadget; similarly, we call it an *output-wire* if it is connected to an output-cell of a gadget.

The *crossing gadget* addresses the fact that edges of G_{2M} can cross in an embedding of the graph. The layout is defined by the 9×9 cells as given in Figure 5.3. It ensures that vertical and horizontal wires can cross in $\square_{2M}$ such that a fire expanding along one wire does not affect the state of the cells along the other wire. Since G_{2M} is not necessarily planar, the embedding of G_{2M} in the geometric interpretation is likely to have (many) crossing edges. Certainly, a crossing of two wires can be easily realized with a cell-graph of three or more layers; however, one would still have to describe how to place the crossing in order to avoid that too many crossings occur at the same spot. In the following



(i) Layout of the cells in $\mathscr{L}_1$ with four input-cells, $\mathbf{I}^{\text{out}}$ in the bottom row, and $\mathbf{I}^{\text{cen}}$ in the centre.



(ii) Layout of the cells in $\mathscr{L}_2$ with four input-cells and one output-cell.

Figure 5.4: Layout of the basic instruction gadget with the initial x- and y-values given in the cells. It is identical to the layout of the specific instruction gadget for subtraction instructions involving r_2. The alive cells of $\mathscr{L}_2$ connect the cells $\mathbf{I}^{\text{in}}$ and $\mathbf{I}^{\text{out}}$ to vertical wires in $\mathscr{L}_2$ outside of the gadget.

we describe how $\square_{2M}$ can be constructed using only two layers. Since crossings cannot be realised in a single layer of cells,[6] two is the minimum number of layers required to make the reduction work in the model as defined in Definition 12. The following lemma captures the essential property of a crossing gadget.

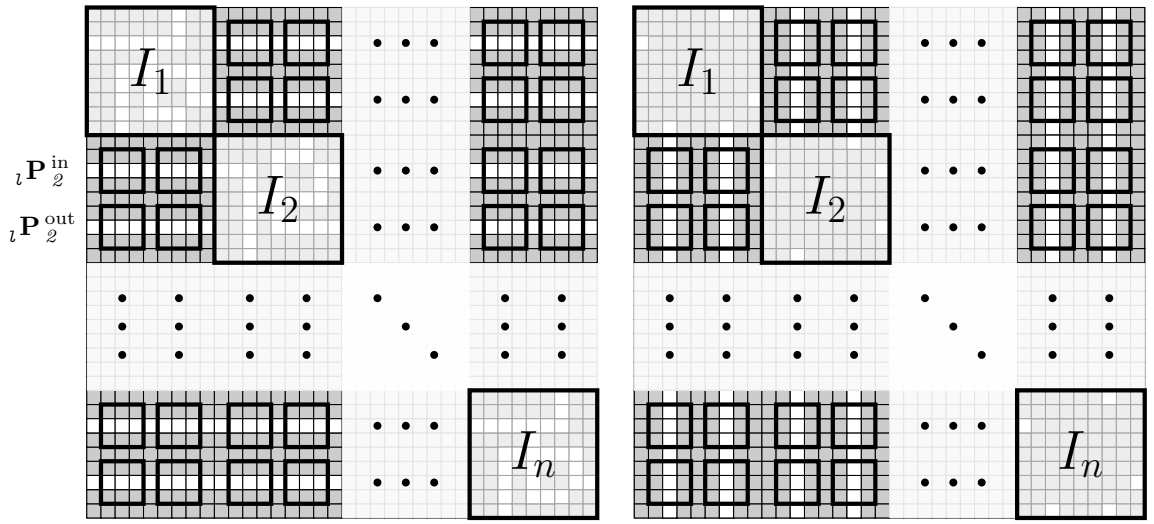
Lemma 19 (crossing gadget). *Let* $\mathbf{X}^{\text{in}}$ *be an input-cell and* $\mathbf{X}^{\text{out}}$ *an output-cell of a crossing gadget connected with a straight vertical or horizontal wire.*

If $\mathbf{X}^{\text{in}}$ *is the first cell of the adjacent wire that ignites at time* t*, the following holds:* $\mathbf{X}^{\text{out}}$ *ignites at time* $t' \leq t + 11$*; the state of all cells that do not belong to the wire remains unchanged; and, at time* $t' + 1$*, none of the gadget's cells are burning.*

Note that the x-y-value pairs of $_l\mathbf{X}^{\text{in}}$ and $_r\mathbf{X}^{\text{out}}$ (or $_r\mathbf{X}^{\text{in}}$ and $_l\mathbf{X}^{\text{out}}$) can be swapped to ensure that both horizontal wires run to the left (right). Similarly, the wires of $\mathscr{L}_2$ can be directed so that both run upwards or downwards. None of these changes invalidates Lemma 19. This will be used in the description of the layout of a connector gadget.

The *instruction gadget* addresses the fact that a vertex v_i of G_{2M} has at most one outgoing edge. The basic layout is defined by the 9×9 cells as given in Figure 5.4. It ensures that once a fire reaches the gadget via one of the four input-cell, it can only expand via the unique output-cell. This functionality is realised by layer $\mathscr{L}_1$ of the gadget; the purpose of $\mathscr{L}_2$ is to connect vertical wires outside of the gadget to $\mathscr{L}_1$. Recall that every vertex v_i of G_{2M} has a finite number of incoming edges and at most one outgoing edge. For the instruction gadget, we assume that v_i has at most a single incoming edge per direction; later, we explain why this can be assumed and how incoming edges of v_i are bunched together with the connector gadget. Thus, the instruction gadget has only a single input-cell into each direction denoted by $_l\mathbf{I}^{\text{in}}$, $_r\mathbf{I}^{\text{in}}$, $_t\mathbf{I}^{\text{in}}$ and $_b\mathbf{I}^{\text{in}}$. Moreover, we call the cell with

[6] The impact of a burning cell is the same for every neighbouring cell; hence, it is impossible to realize a crossing within a single layer avoiding that a fire ignites both wires.



(i) Layout of the cells in $\mathscr{L}_1$. (ii) Layout of the cells in $\mathscr{L}_2$.

Figure 5.5: Layout of the program gadget: An I_i-instruction gadget is placed for each $1 \leq i \leq n$ along the diagonal of a $n \times n$ block matrix; each block consists of 9×9 cells. Into each remaining block, a crossing gadget is placed. This sketch assumes that I_1 is a subtraction instruction involving r_2 and I_2 is an addition instruction involving r_1.

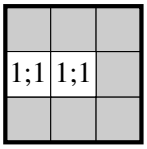
weight $(1;1)$ in the centre of $\mathscr{L}_1$ the *centre-cell* and denote it with $\mathbf{I}^{\text{cen}}$. The following lemma captures the crucial property of an instruction gadget.

Lemma 20 (instruction gadget). *Let $\mathbf{I}^{\text{in}}$ be an input-cell, $\mathbf{I}^{\text{out}}$ be an output-cell, and $\mathbf{I}^{\text{cen}}$ be the centre-cell in $\mathscr{L}_1$ of the basic instruction gadget. If $\mathbf{I}^{\text{in}}$ ignites at time t, $\mathbf{I}^{\text{cen}}$ ignites at time $t' \leq t+8$; if $\mathbf{I}^{\text{cen}}$ ignites at time t, $\mathbf{I}^{\text{out}}$ ignites at time $t'' \leq t+8$. The state of all other input-cells, except for $\mathbf{I}^{\text{in}}$'s direct neighbour, remains unchanged; and, at time $t''+1$, none of the gadget's cells are burning.*

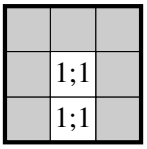
Given the basic instruction gadget, we can define a specific I_i-instruction gadget for every instruction I_i of *2M*. For the halting instruction I_n, we define the I_n-instruction gadget as a copy of the basic variant for which we set the values of $\mathbf{I}^{\text{out}}$ to $(0;0)$ in both layers; although the cells' values are no longer $(2;1)$, we still refer to them as the output-cells of the I_n-instruction gadget. Thus, Lemma 20 still holds for the I_n-instruction gadget, except for the fact that $\mathbf{I}^{\text{out}}$ never ignites. For any other instruction I_i, we define a specific I_i-instruction gadget based on rotations of the basic variant by multiples of 90°: If I_i is an addition instruction involving r_1 (r_2), the basic variant is rotated by 90° (180°); if I_i is a subtraction instruction involving r_1 (r_2), the basic variant is rotated by 270° (0°). Since none of the rotations invalidates Lemma 20, it holds for every specific I_i-instruction gadgets where $I_i \neq I_n$. To indicate that an input-cell belongs to a specific I_i-instruction gadget, we use the subscript index i; as before, the pre-subscript indicates the direction where the cell comes to lie after the rotation: For example, if I_i is an addition instruction to register r_1, the output-cells of the I_i-instruction gadget lie to the right and we denote such a cell with $_{r}\mathbf{I}^{\text{out}}_{i}$.

The *program gadget* addresses the fact that a box in G_{2M} contains multiple vertices which need to be systematically arranged. The layout is defined via $n \times n$ blocks of cells where each block consists of 9×9 cells, see Figure 5.5. For $1 \leq i \leq n$, we place an I_i-instruction gadget into the i-th block along the diagonal of the matrix; into each remaining block, we place a crossing gadget. The layout ensures that incoming and outgoing wires of instruction gadgets intersect only at specific positions and two

Figure 5.6: Layout of the corners with which a crossing of two wires can be replaced. This ensures that a fire burning along a wire on $\mathscr{L}_1$ can switch and continue to burn along another wire in $\mathscr{L}_2$ and vice versa.



(i) Layout of the cells in $\mathscr{L}_1$.



(ii) Layout of the cells in $\mathscr{L}_2$.

layers suffice to model all intersections. For each direction and each $1 \leq i \leq n$, a vertical or horizontal wire connects the input-cell (output-cell) of the I_i-instruction gadget to an input-cell (output-cell) of the program gadget; we denote these cells with ${}_{l}\mathbf{P}_i^{\text{in}}$, ${}_{l}\mathbf{P}_i^{\text{out}}$, ${}_{r}\mathbf{P}_i^{\text{in}}$, ${}_{r}\mathbf{P}_i^{\text{out}}$, ${}_{t}\mathbf{P}_i^{\text{in}}$, ${}_{t}\mathbf{P}_i^{\text{out}}$, ${}_{b}\mathbf{P}_i^{\text{in}}$, ${}_{b}\mathbf{P}_i^{\text{out}}$. The following lemma captures the gadget's property.

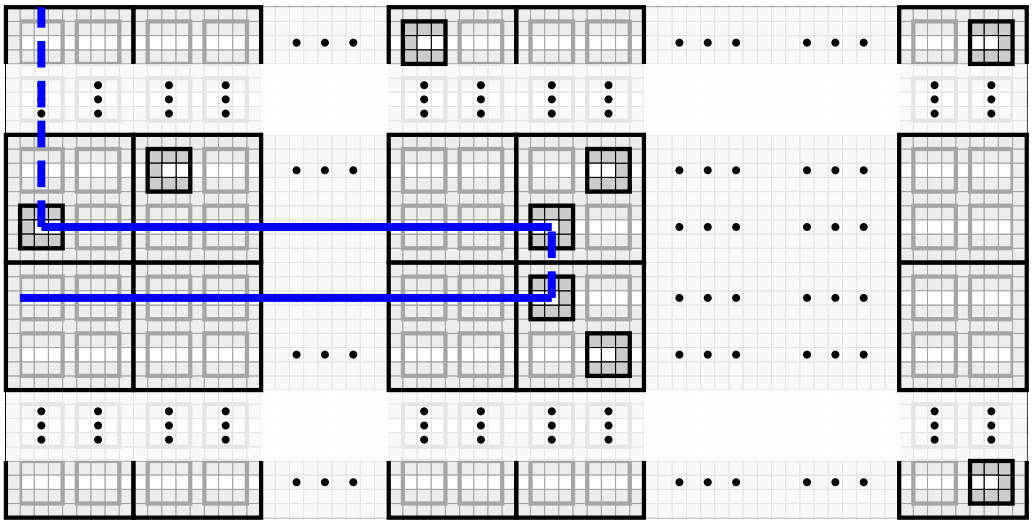
Lemma 21 (program gadget). *Let $\mathbf{P}_i^{\text{in}}$ be an input-cell and $\mathbf{P}_i^{\text{out}}$ be an output-cell of the program gadget which are connected to the I_i-instruction gadget with a horizontal or vertical wire. Moreover, let $\mathbf{I}_i^{\text{cen}}$ be the centre-cell of the I_i-instruction gadget, $T \coloneqq 12(n-1)+8$, and n the number of instructions of 2M.*

If $\mathbf{P}_i^{\text{in}}$ ignites at time t, then $\mathbf{I}_i^{\text{cen}}$ ignites at time $t' \leq t+T$. Moreover, if $\mathbf{I}_i^{\text{cen}}$ ignites at time t, then ${}_{r}\mathbf{P}_i^{\text{out}}$ (${}_{l}\mathbf{P}_i^{\text{out}}$) ignites at time $t'' \leq t+T$ if and only if I_i is an addition (subtraction) instruction for register r_1; and, ${}_{t}\mathbf{P}_i^{\text{out}}$ (${}_{b}\mathbf{P}_i^{\text{out}}$) ignites at time $t'' \leq t+T$ if and only if I_i is an addition (subtraction) instruction for register r_2. The state of all other boundary cells remains unchanged and, at time $t''+1$, none of the gadget's cells are burning.

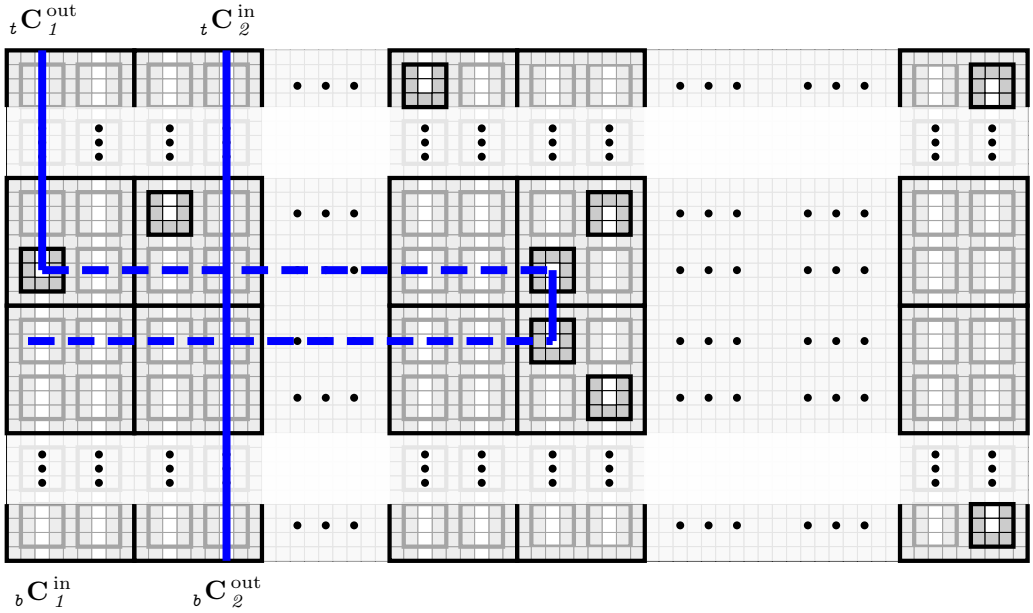
The *connector gadget* addresses two facts: A vertex v_i of G_{2M} can have a large number of incoming edges whereas an instruction gadget expects only a single input-wire per direction; moreover, for $i \neq n$, the outgoing edge connects v_i to a vertex of the same or a neighbouring box. The basic layout is defined via $n \times 2n$ blocks of cells where each block consists of 9×9 cells; into each block we place a crossing gadget. The first n horizontal wires from the top of the crossing gadget are oriented to run from right to left; the first n horizontal wires from the bottom of the gadget are oriented to run from left to right. The orientation of the vertical wires does not change, i. e., for $1 \leq i \leq n$, the $(2i-1)$-th wire from the left runs upwards whereas the $2i$-th wire runs downwards. As indicated in Figure 5.7, we change crossings to *corners* to connect the i-th horizontal wire from below to the i-th horizontal wire from above; similarly, we proceed to connect the i-th vertical wire from the left to the $(n+i)$-th vertical wire from the left. Layers in the layout of a corner, which are given in Figure 5.6, can be swapped and rotated by multiples of 90° to ensure that a fire burning along one wire continues along the other wire only in the desired direction. Thus, the i-th and $(2n+1-i)$-th horizontal wire from below are connected to the i-th output-cell from the left in the top row. Altogether, we call these cells the *output-wire of* ${}_{t}\mathbf{C}_i^{\text{out}}$. This implies the following two facts: First, many output-wires of different instruction gadgets can be linked to the same output-wire of ${}_{t}\mathbf{C}_i^{\text{out}}$. Second, every output-wire of an instruction gadget can pass through the gadget, cf. Figure 5.7. The following lemma captures these two properties.

Lemma 22 (connector gadget). *In a basic connector gadget, let ${}_{t}\mathbf{C}_j^{\text{in}}$, ${}_{b}\mathbf{C}_j^{\text{out}}$ be the input-cell and output-cell of the $(2j)$-th vertical wire from the left. Moreover, let ${}_{t}\mathbf{C}_i^{\text{out}}$ be the output-cell of the $(2i-1)$-th vertical wire from the left and let c be a cell of the attached wire that runs through both layers.*

If ${}_{t}\mathbf{C}_j^{\text{in}}$ ignites at time t, then ${}_{b}\mathbf{C}_j^{\text{out}}$ ignites at time $t' \leq t+12n$. Moreover, if c ignites at time t, then ${}_{t}\mathbf{C}_i^{\text{out}}$ ignites at time $t' \leq 60n$. In both cases, the state of all other boundary cells remains unchanged and, at times $t'+1$ and $t''+1$, none of the gadget's cells are burning.



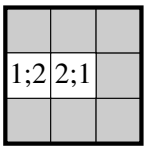
(i) Layout of the cells in $\mathscr{L}_1$.



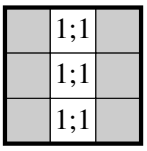
(ii) Layout of the cells in $\mathscr{L}_2$.

Figure 5.7: The layout of a connector gadget consists of $n \times 2n$ block; each block consist of 9×9 cells and contains a crossing gadget. All boundary cells, except for the ones at the top edge in $\mathscr{L}_2$ are replaced with dead cells. The output-wire of ${}_t\mathbf{C}_1^{\text{out}}$ is indicated with a blue path: the vertical, dashed lines in (i) indicate where the wire runs in $\mathscr{L}_2$; similarly, horizontal, dashed lines in (ii) indicate where the wire runs in $\mathscr{L}_1$. Note that the left $n \times n$ blocks have input-cells and output-cells in the bottom row while the $n \times n$ blocks on the right have only dead boundary cells. Further note that the input-wire of ${}_b\mathbf{C}_i^{\text{in}}$ is disconnected from the output-wire of ${}_t\mathbf{C}_j^{\text{out}}$ for every $1 \leq i, j \leq n$.

Figure 5.8: Layout of the junctions with which a crossing of two wires can be replaced. This ensures that an input-wire of the connector gadget can be connected to an output-wire.



(i) Layout of the cells in $\mathscr{L}_1$.



(ii) Layout of the cells in $\mathscr{L}_2$.

Given the basic connector gadget, we can define specific variants for each direction. The variants are obtained by rotations of the gadget by multiples of 90°: A $\mathtt{b}$-connector ($\mathtt{t}$-connector) gadget is obtained from the basic variant by a rotation of 0° (180°); an $\mathtt{r}$-connector ($\mathtt{l}$-connector) gadget is obtained from the basic variant by a rotation of 90° (270°) and a swap of the layers. Note that neither rotating the gadget nor swapping the layers invalidates Lemma 22.

The basic *box gadget* gives a full description of how vertices and edges of a single box in G_{2M} are mapped to cells of $\square_{2M}$. The basic layout is defined via 3×3 blocks of cells where each block consists of $9n \times 9n$ cells, which is exactly the size of a program gadget. A program gadget is placed in the center block; the remaining blocks are filled with specific connector gadgets: A $\mathtt{l}/\mathtt{r}/\mathtt{b}/\mathtt{t}$-connector gadget is placed left/right/above/below of the program gadget such that the output-cells of the connector gadget are adjacent to the input-cells of the program gadget, cf. Figure 5.9.

Depending on the coordinates (b_1, b_2) of a box, we can finally define the layout of a specific box gadget. For this purpose, we modify the basic layout by replacing crossings with *junctions*; thus, we connect input-cells of the box gadget (or output-cells of program gadget in the center) to the output-wires of the connector gadgets placed around the center. Layers in the layout of a junction, which are given in Figure 5.8, can be swapped and rotated by multiples of 90° to ensure that a fire burning along one wire continues along the output-wire of the connector gadget into the desired direction[7] . This is done according to the following rules:

- If $I_i = (i, r, j)$ is an addition instruction with $r = r_1$ ($r = r_2$), then connect ${}_{\mathtt{l}}\mathbf{B}_i^{\text{in}}$ (${}_{\mathtt{b}}\mathbf{B}_i^{\text{in}}$) to the j-th output-wire of the $\mathtt{l}$-connector ($\mathtt{b}$-connector) gadget;

- if $I_i = (i, r, j, k)$ is a subtraction instruction with $r = r_1$ and $b_1 > 0$ ($r = r_2$ and $b_2 > 0$), then connect ${}_{\mathtt{r}}\mathbf{B}_i^{\text{in}}$ (${}_{\mathtt{t}}\mathbf{B}_i^{\text{in}}$) to the k-th output-wire of the $\mathtt{r}$-connector ($\mathtt{t}$-connector) gadget;

- if $I_i = (i, r, j, k)$ is a subtraction instruction with $r = r_1$ and $b_1 = 0$ ($r = r_2$ and $b_2 = 0$), then connect ${}_{\mathtt{l}}\mathbf{P}_i^{\text{out}}$ (${}_{\mathtt{b}}\mathbf{P}_i^{\text{out}}$) to the j-th output-wire of the $\mathtt{l}$-connector ($\mathtt{b}$-connector) gadget.

Figure 5.9 exemplifies these modifications. All specific box gadgets together[8] give a complete description of $\square_{2M}$ where cells of the cell-graph that do not belong to a box are filled with dead cells. The following lemma ensures that the connections meet the intended purpose.

[7] Note that, by construction, the output-wire is directed which ensures that fire can only expand into one direction.
[8] Note that there are only four different ones.

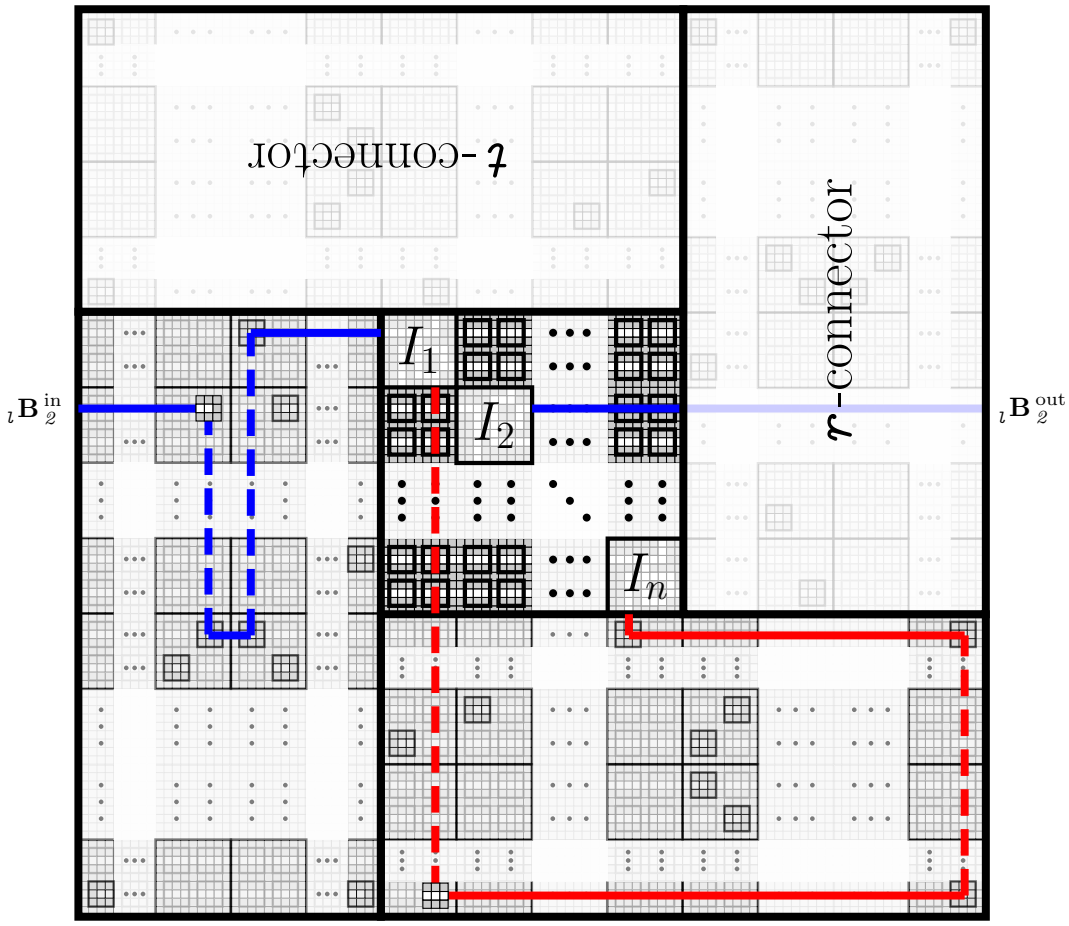


Figure 5.9: The layout of $\mathscr{L}_1$ of a box gadget. For the example, we assume that coordinate $b_2 = 0$, $I_2 = (2, r_1, 1)$ is an addition instruction (blue lines), and $I_1 = (1, r_2, n, 2)$ is a subtraction instruction (red lines). As before, dashed lines indicate where wires continue in $\mathscr{L}_2$. Junctions have been placed to connect ${}_{\mathfrak{l}}\mathbf{B}_2^{\text{in}}$ to the first output-wire of the $\mathfrak{l}$-connector gadget and ${}_{\mathfrak{b}}\mathbf{P}_I^{\text{out}}$ to the n-th output-wire of the $\mathfrak{b}$-connector gadget.

Lemma 23 (box gadget). *Let ${}^{(b_1,b_2)}\mathbf{I}_i^{\text{cen}}$ be the centre-cell of the I_i-instruction gadget within the box gadget of coordinates (b_1,b_2) where $I_i \neq I_n$.*

If ${}^{(b_1,b_2)}\mathbf{I}_i^{\text{cen}}$ ignites at time t, exactly one of the following implications holds:

- ${}^{(b_1+1,b_2)}\mathbf{I}_j^{\text{cen}}$ *ignites at time t' if $I_i = (i, r_1, j)$;*
- ${}^{(b_1,b_2+1)}\mathbf{I}_j^{\text{cen}}$ *ignites at time t' if $I_i = (i, r_2, j)$;*
- ${}^{(b_1-1,b_2)}\mathbf{I}_k^{\text{cen}}$ *ignites at time t' if $I_i = (i, r_1, j, k)$ and $b_1 > 0$;*
- ${}^{(b_1,b_2-1)}\mathbf{I}_k^{\text{cen}}$ *ignites at time t' if $I_i = (i, r_2, j, k)$ and $b_2 > 0$;*
- ${}^{(b_1,b_2)}\mathbf{I}_j^{\text{cen}}$ *ignites at time t' if $I_i = (i, r_1, j, k)$ and $b_1 = 0$;*
- ${}^{(b_1,b_2)}\mathbf{I}_j^{\text{cen}}$ *ignites at time t' if $I_i = (i, r_2, j, k)$ and $b_2 = 0$;*

where t' is bounded by a constant depending on n, the number of instructions of 2M.

Using Lemma 23, we finally prove the following theorem. Note that neither the following theorem nor its proof depend on G_{2M}; in exchange, we can bound the simulation time by means of instructions of *2M*. The proof of the theorem is similar to the one for Observation 5. The idea is to employ induction over the number of transitions of *2M* for the forward direction; for the reverse direction, we employ induction over the discrete number of simulation steps.

Theorem 11. *The fire-expansion model can simulate every two-register machine 2M with constant time overhead, where the constant depends on the number of instructions of 2M.*

Proof. We prove the following equivalence: When *2M* is started in configuration (b_1,b_2,I_i), it reaches configuration (b_1^*,b_2^*,I_k) within m steps if and only if there exists some $t_m \geq 0$ such that, in the instance $\square_{2M}$ of the fire-expansion model, ${}^{(b_1^*,b_2^*)}\mathbf{I}_k^{\text{cen}}$ ignites at time t_m if cell ${}^{(b_1,b_2)}\mathbf{I}_i^{\text{cen}}$ is initially set on fire; i. e., the x-value of ${}^{(b_1,b_2)}\mathbf{I}_i^{\text{cen}}$ is set to zero at time $t = 0$.

We prove the forward direction via induction over m: If $m = 0$, the hypothesis holds since only one cell is set on fire at $t_0 = 0$. For the induction step $(m-1) \to m$, we may assume that the last transition of *2M* is from configuration $C_{m-1} = (b_1', b_2', j)$ to $C_m = (b_1^*, b_2^*, k)$; hence I_j is not the halting instruction. By induction hypothesis, ${}^{(b_1',b_2')}\mathbf{I}_j^{\text{cen}}$ ignites at some time t_{m-1} if cell ${}^{(b_1,b_2)}\mathbf{I}_i^{\text{cen}}$ is set on fire at time $t = 0$. Since C_{m-1} and C_m are successive two-register machine configurations, it is either $b_1' = b_1^*$ and $|b_2' - b_2^*| \leq 1$, or $b_2' = b_2^*$ and $|b_1' - b_1^*| \leq 1$. Thus, ${}^{(b_1',b_2')}\mathbf{I}_j^{\text{cen}}$ and ${}^{(b_1^*,b_2^*)}\mathbf{I}_k^{\text{cen}}$ lie either in the same or in neighbouring box gadgets. Since I_j is not the halting instruction, by Lemma 23, ${}^{(b_1^*,b_2^*)}\mathbf{I}_k^{\text{cen}}$ ignites at time $t_m \leq t_{m-1} + t'$. Since, t' is linear in the number of instructions n, the time overhead is constant per simulation step.

We prove the reverse direction via induction over t_m. Provided that ${}^{(b_1^*,b_2^*)}\mathbf{I}_k^{\text{cen}}$ ignites at time t_m when ${}^{(b_1,b_2)}\mathbf{I}_i^{\text{cen}}$ is initially set on fire, we have to show that *2M* reaches configuration (b_1^*,b_2^*,I_k) when started in (b_1,b_2,I_i). For the induction start $t_m = 0$, the only cell in the cell-graph which is burning at t_m is ${}^{(b_1,b_2)}\mathbf{I}_i^{\text{cen}}$ by assumption. Thus, ${}^{(b_1^*,b_2^*)}\mathbf{I}_k^{\text{cen}}$ and ${}^{(b_1,b_2)}\mathbf{I}_i^{\text{cen}}$ are the same cells and the hypothesis holds, since *2M* obviously reaches its initial configuration without a single transition.

For the induction step, we know that $t_m > 0$. Consequently, ${}^{(b_1,b_2)}\mathbf{I}_i^{\text{cen}}$, ${}^{(b_1^*,b_2^*)}\mathbf{I}_k^{\text{cen}}$ have to be different cells. Let ${}^{(b_1',b_2')}\mathbf{I}_j^{\text{cen}}$ denote the centre-cell which ignited at time $t_{m-1} < t_m$ right before ${}^{(b_1^*,b_2^*)}\mathbf{I}_k^{\text{cen}}$; i. e., no centre-cell in the cell-graph ignites between t_{m-1} and t_m.[9] Note that, at time

[9] Note that ${}^{(b_1',b_2')}\mathbf{I}_j^{\text{cen}} = {}^{(b_1,b_2)}\mathbf{I}_i^{\text{cen}}$ is possible.

t_{m-1}, all burning cells in the cell-graph lie within the instruction gadget of ${}^{(b'_1,b'_2)}\mathbf{I}_j^{\text{cen}}$; moreover, by Lemma 23, ${}^{(b'_1,b'_2)}\mathbf{I}_j^{\text{cen}}$ and ${}^{(b^*_1,b^*_2)}\mathbf{I}_k^{\text{cen}}$ have to lie in the same or in neighbouring box gadgets.[10] Additionally, Lemma 23 shows that when *2M* is in configuration (b'_1,b'_2,I_j), it proceeds to configuration (b^*_1,b^*_2,I_k) in a single step. By induction hypothesis, *2M* reaches configuration (b'_1,b'_2,j) within $m-1$ steps when started in configuration (b_1,b_2,I_i). Altogether, *2M* reaches configuration (b^*_1,b^*_2,I_k) within m steps when started in configuration (b_1,b_2,I_i), which completes the proof. □

Implications of the Proof

Compared to simulations of Turing machines in other models, dead cells cannot regenerate their state during the simulation in the fire-expansion model. Once a cell reaches the state dead during the fire-expansion, the cell remains in this state until the simulation ends. This makes it necessary to encode information about *2M*'s state in the geometric information of $\square_{2M}$'s cell-graph; more precisely, the state of *2M* corresponds to the state of some specific cells in the embedded cell-graph and vice versa.

The proof of Theorem 11 implies that many interesting and apparently simple questions cannot be algorithmically solved in general. By reduction from the halting problem for Turing machines, we obtain the following corollary.

Corollary 10. *There is no algorithm that can decide for every instance of the fire-expansion model and two cells c,c' whether cell c' eventually ignites when cell c is initially set on fire.*

The reduction and proof of Theorem 11 shows that it is even impossible to decide whether the fire comes close to c'.

Corollary 11. *Given an instance of the fire-expansion model, two cells c,c' and a finite radius $r \in \mathbb{N}$.*

There is no algorithm that can decide whether there is a cell within distance[11] *less that r from c' which eventually ignites when cell c is initially set on fire.*

The proof of the corollary can be sketched as follows. In the construction of $\square_{2M}$, we required $27n \times 27n$ cells to realize a box gadget; certainly, we can realize the box gadget within any larger number of cells, e. g., for $27n \cdot r \times 27n \cdot r$ cells with instruction gadgets of $9r \times 9r$ cells. Now assume, to the contrary, the problem was decidable for some (possibly) large, constant radius r. Thus, however, we could decide the problem of Corollary 10: If c'' lies close to c', a transition of c'' form alive to burning already implies that c' eventually ignites; this, however, gives a contradiction.

Many other interesting questions, e. g., whether a fire extinguishes over time without the need to fight it, are algorithmically unsolvable as well.

Finally, the proof shows that a cellular automaton with $d = 3$, $k = 6$ and the $r = 1$ Moore neighbourhood can simulate every Turing machines if the update rule forbids rebirth of cells. Similar to the fire-expansion model, the cellular automaton requires only two layers, since cells cannot restore their initial states. The number of states $k = 6$, stems from the fact that only four different types of x-y-value pairs have been used in the cells; two additional pairs occur when counter values diminish during the simulation.

[10] Assume to the contrary this were not true, then the fire has to pass through another box gadget to reach ${}^{(b^*_1,b^*_2)}\mathbf{I}_k^{\text{cen}}$. This, however, is only possible when passing through another instruction gadget which contradicts the assumption that ${}^{(b'_1,b'_2)}\mathbf{I}_j^{\text{cen}}$ ignited right before ${}^{(b^*_1,b^*_2)}\mathbf{I}_k^{\text{cen}}$.

[11] Here, the distance between two cells can be defined e. g. via the L_1 metric.

5.4 Conclusion and Outlook

In this chapter, we have introduced a new, discrete model for the study of fire-expansion and fire-fighting problems in Theoretical Computer Science. The model generalizes existing models by adding two integer parameters to each cell of the cell-graph. This simple extension added a lot of computational power to the model: We proved that with a cell-graph of only two layers, it is possible to simulate the computations of arbitrary Turing machines. This implies that already many interesting fire-expansion problems in this general model are algorithmically undecidable. Interesting variants of this model can modify the cell-graph or the update rule (transition function).

Modifications of the cell-graph could use cells of a different shape, restrict the size of the cell-graph and even the values stored in the cells. Instead square cells, one could use a graph where each layer consist, e. g., of hexagonal cells. In general, one could use any other regular cell-graphs and the results of this chapter still apply for this case. If the size of the cell-graph is restricted to a finite number of cells in a single layer, fire-expansion and also fire-fighting problems immediately become decidable. Since many fire-fighting questions still turn out to be hard to solve, one can further impose restrictions to the x- and y-values stored in the cells. Yet, there are still many interesting, but difficult problems to study. Some initial results are presented in [66, 67] and the doctoral thesis of Barbara Schwarzwald [90].

Modifications of the update rule could include the following ones. On the one hand, additional parameters can be introduced to describe weather conditions as well as crown or ground fires; these extensions could be applied to single cells individually or uniformly to the entire lattice. On the other hand, the update rule could allow cells to restore their initial x- and y-values over time. Thus, they can change their state from dead back to alive. Note that this modification makes it possible to realize a crossing of wires even within a single layer [66, 67] which implies for this variant that Theorem 11 already holds for a cell-graph which with a single layer of cells. Note that in the model as introduced in this thesis, such a transition from dead to alive is impossible. This is also a major difference to other Turing universal CA, e. g., GoL, Rule 110, or Wireworld as discussed in Section 5.2.

It will be interesting to further examine the relation between the fire-expansion model and popular CAs such as GoL, Rule 110, and Wireworld. One may ask whether the fire-expansion model can be used to simulate other popular CAs such GoL, Rule 110, or Wireworld. A major difference appears to be the transition function: In GoL cells can die from overpopulation for which there is no equivalent transition in the fire-expansion model; similarly, dead cells in GoL can be reborn which is impossible in the fire-expansion model. Moreover, in some CAs, such as GoL or Wireworld, one can simulate the logical gadgets $\wedge$, $\vee$ and $\neg$. It would be highly interesting to see whether this is also possible with (a variant) of the fire-expansion model. One may further ask whether GoL, Rule 110, or Wireworld can be simulated with the fire-expansion model. This is not trivial since the transition graph of a cell in these automata is cyclic, whereas it is acyclic for the fire-expansion model. For many CAs this property is essential for the simulation of Turing machines.

The relation between arbitrary CAs and the fire-expansion model gives rise to several questions. In Section 5.2, we stated that CAs can simulate the fire-expansion model in linear time; however, the construction requires a certain (potentially large) number of states. Can this simulation be done more efficiently, i. e., avoiding the large increase in the number of states? Conversely, can the fire-expansion model simulate every CA (in linear time)? If not, which modification of the fire-expansion model makes both models computationally equivalent? Similarly, it remains open whether NN can be used to simulate the fire-expansion model efficiently, i. e., to avoid that the number of states of the NN depend on the size of the x-y-values stored in the cells. It is also conceivable that y-values can be used

as edge-weights of the NN to achieve a simulation of the fire-expansion by a NN in sublinear time. Again, (under which extensions) does the converse hold?

Chapter 6

Conclusion and Research Perspectives

In four chapters, this thesis examined different search problems which are related to or emerge from a geometric context. Each problem was analysed in a new model or context where surprising and non-trivial results were proven. This does not only add a certain twist to these problems, it further leads to a better understanding of the problems and helps to provide mathematical explanations for related problems and general observations. In the following, we briefly summarise the results, discuss implications at a meta-level and suggest directions for further research.

Chapter 2 examined the problem of searching for a stationary target point in a bounded part of the Euclidean space. The chapter considered a new query model where feedback was given by an ordering of the query points. With regard to the feedback, we distinguished two variants of the model. For both variants, we proved lower and upper bounds for the precision that can be achieved with a certain number of queries. However, for both variants, a gap between lower and upper bound remained: In the one-shot variant, the gap is only a constant factor, even in the two-dimensional setting; in contrast to this, the gap grows exponentially in the incremental variant, already in the one-dimensional setting. It would certainly be interesting to see how the results can be improved to narrow these gaps.

Additionally, finding a different query model which allows for substantial improvements in comparison to binary searching strategies, would be fascinating. In the incremental variant, both strategies, the new query strategy and the binary search approach, achieved an accuracy which grows single exponentially with n, the number of queries; namely, $\mathrm{O}(2.29^n)$ and $\mathrm{O}(2^n)$, respectively. Furthermore, no such query strategy can achieve an accuracy which grows faster than $\mathrm{O}(3.66^n)$. Generally speaking, this implies that, even in this stronger query model, the accuracy of every query strategy is limited to single exponential growth with a rather small, constant base. As discussed in Section 2.5, this shows that one cannot do fundamentally better than binary searching. However, is there a different query model whereby a search strategy can outperform the binary search approach by an order of magnitude? That is to say, while the accuracy of the binary search approach in this model still grows single exponentially, could another strategy achieve accuracy with, e. g., doubly exponential growth? Such results might shed new light on the conditions under which binary searching is essentially optimal; or, conversely, allow for a better insight and understanding of the conditions under which query strategies can be substantially better than a binary search approach.

Chapter 3 examined the search problem for a boundary point of a region in the Euclidean plane, which is related to Bellman's escape path problem. We introduced and motivated a new measure to capture the intrinsic complexity of such geometric search problems. It provided the basis for our analysis of search strategies for the initial problem. We developed lower and upper bounds on the approximation factor that can be achieved and discussed what this implies for Bellman's escape path

problem. Generally speaking, this offered an alternative for proving the optimality of geometric search strategies, which we exemplified for two spiral search strategies. The results provided evidence for the general observation that spiral searching is highly efficient for many geometric search problems in the Euclidean plane. This seems to stem from the fact that spiral trajectories balance a BFS and DFS approach by increasing the search depth exponentially into every direction. In general, increasing the search depth exponentially is a design paradigm for search strategies, cf. [65]. For several other search problems, this paradigm gave optimal search strategies; see, e. g., [3, 8]. It would be interesting to see how our approach can be applied to other geometric search problems in order to prove the optimality of strategies which have not been improved for many years. For example, for the problem of searching for a line in the plane, optimality of a certain spiral strategy is a long-standing conjecture [35, 37].

Particularly fascinating would be to see further progress on the shortest escape path problem. The problem has attracted a lot of attention and could only be solved for special types of regions up to this date. Finding a shortest escape path for a given region is tedious work; for this reason, we decided to seek for approximations of shortest escape paths with a small, constant factor. We provided such an approximation for the family of regions which consist of all fat, convex shapes, all equilateral triangles, and all infinite strips. The missing key to link our approximation results to shortest escape paths for a larger family of regions seems to be the following claim: For every region with a non-empty kernel[1], the certificate path approximates a shortest escape path with a small constant factor. Note that the certificate path is actually well-defined for every path-connected region. Although it seems to be a reasonable measure for the complexity of escape problems in every path-connected region, it is impossible to approximate it: While the certificate path can locate a single point in the plane, this is impossible for a trajectory that approximates the certificate path with a constant factor. Hence, for the family of path-connected regions, it is certainly necessary to develop different measures for the inherent complexity of geometric search problems.

Chapter 4 examined the search for an ε-rounding on an edge-weighted graph which meets three constraints. The first constraint ensures that the absolute change in weight along each shortest path is less than ε. The second constraint ensures that a shortest path remains shortest after rounding the edge-weights. Unlike other authors, we imposed the third, new constraint that a shortest path, after the rounding, had to have already been shortest beforehand. We discussed how the difficulty of this problem depends on the value of ε. By reduction from 3-SAT, we proved that, even in its simplest form, the problem is NP-hard for $\varepsilon \in (3/4; 1]$. However, in the case where the graph is a tree, we showed how to compute a solution efficiently. The fact that NP-hard problems become polynomial-time solvable on restricted graph classes is not surprising; the same effect can be observed for many other graph problems, e. g., vertex cover or dominating set. But where exactly does the difficulty of this rounding problem lie? It will be most interesting to see whether the problem can be efficiently solved for other graph classes such as cycles, DAGs and planar graphs. Especially with respect to planar graphs, the question is most urgent since graphs are (almost) planar in many practical applications.

Although it is reasonable to assemble the constraints as done in Definition 10, it makes sense to consider them separately. It will be interesting to see whether path-oblivious ε-roundings with rather small, constant values of ε exist also for more general graph classes and can be computed efficiently; recall that, a 1-rounding (2-rounding) can be computed efficiently for the special case where the graph is a path (tree). Do similar approximations also exist for other graph classes? With regard to the second and third constraint, it is not even clear whether both of them can be ensured at

[1] Maybe even for every simply connected region.

the same time for every graph. The discussion in Section 4.1 suggests that this is impossible when each constraint presupposes the prior one. Thus, it would be necessary to further study the interplay of these conditions, assemble them in a different manner or even consider them independent of each other.

Chapter 5 examined the search for specific configurations in a new, two-dimensional fire-fighting model. The new model generalises existing ones by adding two integer parameters to each cell which model the cell's resistance against ignition and its remaining fuel. This simple extension adds a lot of computational power to this model, even though, cells can never restore their initial state. In a cell-graph with a single layer and a bounded number of cells, computational problems can always be solved; however, many apparently simple questions seem to resist *efficient* solutions. Efficient solutions have only been found for some few, interesting problems [67, 90]. In a cell-graph where two layers are stacked on top of each other, the model becomes Turing-universal as proven and discussed in this thesis. Thus, for this variant of the model, many interesting problems are inherently unsolvable by means of computers; among them, the problem of searching for a specific configuration in a fire-expansion, e. g., where a specific cell is on fire.

Finally, the results of Chapter 5 take us back to the beginning of this thesis where we considered two computational tasks, a geometric and an arithmetic one. Both of these tasks can be solved, in principle, by Turing machines which can be simulated within the fire-expansion model themselves. Who might have expected that these computations can take place even in this simple model? Please note that we did not address the question as to whether this is also *efficient*. To simulate computations and algorithms efficiently, the eminent challenge seems to be to figure out how to 'program' by means of fire-expansion; this further involves to provide a meaningful encoding and decoding of input and output. In retrospect, the fact that Turing machines can be simulated within the fire-expansion model does not emerge entirely unexpected. Similar phenomena have already been observed for related concepts like cellular automata and artificial neural networks. Therefore, the results should rather be seen as additional evidence for a fundamental observation in nature and science: Simple, local structures with a simple, local behaviour can often lead to incredibly complex global behaviour.

References

[1] Adhikari, A. and Pitman, J. (1989). The shortest planar arc of width 1. *The American Mathematical Monthly*, 96(4):309–327.

[2] Ahn, H.-K., Cheng, S.-W., Cheong, O., Golin, M., and Van Oostrum, R. (2004). Competitive facility location: the voronoi game. *Theoretical Computer Science*, 310(1-3):457–467.

[3] Alpern, S. and Gal, S. (2003). *The theory of search games and rendezvous*, volume 55 of *International series in operations research and management science*. Kluwer.

[4] Arora, S. and Barak, B. (2009). *Computational Complexity - A Modern Approach*. Cambridge University Press.

[5] Asano, T., Katoh, N., Tamaki, H., and Tokuyama, T. (2004). The structure and number of global roundings of a graph. *Theoretical Computer Science*, 325(3):425–437.

[6] Asano, T., Matsui, T., and Tokuyama, T. (2000a). On the complexities of the optimal rounding problems of sequences and matrices. In *Algorithm Theory - SWAT 2000, 7th Scandinavian Workshop on Algorithm Theory, Bergen, Norway, July 5-7, 2000, Proceedings*, pages 476–489.

[7] Asano, T., Matsui, T., and Tokuyama, T. (2000b). Optimal roundings of sequences and matrices. *Nordic Journal of Computing*, 7(3):241–256.

[8] Baezayates, R. A., Culberson, J. C., and Rawlins, G. J. (1993). Searching in the plane. *Information and Computation*, 106(2):234–252.

[9] Baldwin, J. (2004). *The Bulletin of Symbolic Logic*, 10(1):112–114.

[10] Baldwin, J. T. and Shelah, S. (2000). On the classifiability of cellular automata. *Theoretical Computer Science*, 230(1-2):117–129.

[11] Bellman, R. (1956). Minimization problem. *Bulletin of the American Mathematical Society*, 62(3):270.

[12] Besicovitch, A. (1965). On arcs that cannot be covered by an open equilateral triangle of side 1. *The Mathematical Gazette*, 49(369):286–288.

[13] Bloom, G. S. and Golomb, S. W. (1978). Numbered complete graphs, unusual rulers, and assorted applications. In Alavi, Y. and Lick, D. R., editors, *Theory and Applications of Graphs: Proceedings, Michigan May 11–15, 1976*, pages 53–65, Berlin, Heidelberg. Springer Berlin Heidelberg.

[14] Blum, N. (2001). *Theoretische Informatik: Eine anwendungsorientierte Einführung, 2. Auflage*. Oldenbourg.

[15] Börger, E., Grädel, E., and Gurevich, Y. (1997). *The Classical Decision Problem*. Perspectives in Mathematical Logic. Springer.

[16] Braga, G., Cattaneo, G., Flocchini, P., and Vogliotti, C. Q. (1995). Pattern growth in elementary cellular automata. *Theoretical Computer Science*, 145(1-2):1–26.

[17] Bressan, A. (2007). Differential inclusions and the control of forest fires. *Journal of Differential Equations*, 243(2):179–207.

[18] Canny, J. F. and Reif, J. H. (1987). New lower bound techniques for robot motion planning problems. In *28th Annual Symposium on Foundations of Computer Science, Los Angeles, California, USA, 27-29 October 1987*, pages 49–60. IEEE Computer Society.

[19] Chan, T. M., Golynski, A., López-Ortiz, A., and Quimper, C. (2003). Curves of width one and the river shore problem. In *Proceedings of the 15th Canadian Conference on Computational Geometry, CCCG'03, Halifax, Canada, August 11-13, 2003*, pages 73–75.

[20] Cilleruelo, J., Ruzsa, I., and Vinuesa, C. (2010). Generalized sidon sets. *Advances in Mathematics*, 225(5):2786 – 2807.

[21] Cole, R. and Yap, C. K. (1987). Shape from probing. *Journal of Algorithms*, 8(1):19–38.

[22] Cook, M. (2004). Universality in elementary cellular automata. *Complex Systems*, 15(1).

[23] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to Algorithms, 3rd Edition*. MIT Press.

[24] Coulton, P. and Movshovich, Y. (2006). Besicovitch triangles cover unit arcs. *Geometriae Dedicata*, 123(1):79–88.

[25] Culik II, K. and Yu, S. (1988). Undecidability of ca classification schemes. *Complex Systems*, 2(2):177–190.

[26] De Berg, M., Van Kreveld, M., Overmars, M., and Schwarzkopf, O. (1997). *Computational Geometry: Algorithms and Applications*. Springer.

[27] Develin, M. and Hartke, S. G. (2007). Fire containment in grids of dimension three and higher. *Discrete Applied Mathematics*, 155(17):2257–2268.

[28] Dewdney, A. (1990). Computer recreations: The cellular automata programs that create wireworld, rugworld and other diversions. *Scientific American*, 262(1):146–149.

[29] Duncan, C. A., Kobourov, S. G., and Kumar, V. S. A. (2006). Optimal constrained graph exploration. *ACM Trans. Algorithms*, 2(3):380–402.

[30] Erdös, P. and Turán, P. (1936). On some sequences of integers. *Journal of the London Mathematical Society*, s1-11(4):261–264.

[31] Erdös, P. and Turán, P. (1941). On a problem of sidon in additive number theory, and on some related problems. *Journal of the London Mathematical Society*, s1-16(4):212–215.

[32] Eubeler, A., Fleischer, R., Kamphans, T., Klein, R., Langetepe, E., and Trippen, G. (2007). Competitive online searching for a ray in the plane. In *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

[33] Fiat, A. and Woeginger, G. J., editors (1998). *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.

[34] Finbow, S. and MacGillivray, G. (2009). The firefighter problem: a survey of results, directions and questions. *Australasian Journal of Combinatorics*, 43:57–77.

[35] Finch, S. R. (2016). The logarithmic spiral conjecture. *arXiv preprint math/0501133v2*.

[36] Finch, S. R. and Wetzel, J. E. (2004). Lost in a forest. *The American Mathematical Monthly*, 111(8):645–654.

[37] Finch, S. R. and Zhu, L.-Y. (2005). Searching for a shoreline. *arXiv preprint math/0501123*.

[38] Fleischer, R., Kamphans, T., Klein, R., Langetepe, E., and Trippen, G. (2008). Competitive online approximation of the optimal search ratio. *SIAM Journal on Computing*, 38(3):881–898.

[39] Fomin, F. V., Heggernes, P., and van Leeuwen, E. J. (2016). The firefighter problem on graph classes. *Theoretical Computer Science*, 613:38–50.

[40] Funke, S. and Storandt, S. (2016). Consistent rounding of edge weights in graphs. In *Proceedings of the Ninth Annual Symposium on Combinatorial Search, SOCS 2016, Tarrytown, NY, USA, July 6-8, 2016.*, pages 28–35.

[41] Gal, S. (1980). *Search Games*, volume 149. Academic Press.

[42] Gardner, M. (1970). Mathematical games: The fantastic combinations of john conway's new solitaire game "life". *Scientific American*, 223:120–123.

[43] Garzon, M. H. (1995). *Models of Massive Parallelism - Analysis of Cellular Automata and Neural Networks*. Texts in Theoretical Computer Science. An EATCS Series. Springer.

[44] Graham, R. L., Knuth, D. E., and Patashnik, O. (1994). *Concrete Mathematics: A Foundation for Computer Science, 2nd Ed*. Addison-Wesley.

[45] Gray, L. (2003). A mathematician looks at wolfram's new kind of science. *Notices-American Mathematical Society*, 50(2):200–211.

[46] Guibas, L. J. and Hershberger, J. (1989). Optimal shortest path queries in a simple polygon. *Journal of Computer and System Sciences*, 39(2):126–152.

[47] Hagerup, T. (2000). Improved shortest paths on the word RAM. In Montanari, U., Rolim, J. D. P., and Welzl, E., editors, *Automata, Languages and Programming, 27th International Colloquium, ICALP 2000, Geneva, Switzerland, July 9-15, 2000, Proceedings*, volume 1853 of *Lecture Notes in Computer Science*, pages 61–72. Springer.

[48] Haverkort, H. (2014). Embedding cues about travel time in schematic maps. In *Schematic Mapping Workshop*.

[49] Haverkort, H. J., Kübel, D., and Langetepe, E. (2019a). Shortest-path-preserving rounding. In *Combinatorial Algorithms - 30th International Workshop, IWOCA 2019, Pisa, Italy, July 23-25, 2019, Proceedings*, pages 265–277.

[50] Haverkort, H. J., Kübel, D., and Langetepe, E. (2019b). Shortest-path-preserving rounding. *CoRR*, abs/1905.08621.

[51] Haverkort, H. J., Kübel, D., Langetepe, E., and Schwarzwald, B. (2017a). How to play hot and cold on a line. In *Proceedings of the 33rd European Workshop on Computational Geometry, EuroCG 2017, Malmö, Sweden, April 04-07, 2017, Proceedings*, pages 33–36.

[52] Haverkort, H. J., Kübel, D., Langetepe, E., and Schwarzwald, B. (2017b). How to play hot and cold on a line. In *Algorithms and Data Structures - 15th International Symposium, WADS 2017, St. John's, NL, Canada, July 31 - August 2, 2017, Proceedings*, pages 449–460.

[53] Haverkort, H. J., Kübel, D., Langetepe, E., and Schwarzwald, B. (2020). How to play hot and cold. *Computational Geometry*, 87:101596.

[54] Hershberger, J. and Suri, S. (1999). An optimal algorithm for euclidean shortest paths in the plane. *SIAM Journal on Computing*, 28(6):2215–2256.

[55] Hurd, L. P. (1987). Formal language characterizations of cellular automaton limit sets. *Complex Systems*, 1(1):69–80.

[56] Icking, C., Kamphans, T., Klein, R., and Langetepe, E. (2000). On the competitive complexity of navigation tasks. In Hager, G. D., Christensen, H. I., Bunke, H., and Klein, R., editors, *Sensor Based Intelligent Robots, International Workshop, Dagstuhl Castle, Germany, October 15-20, 2000, Selected Revised Papers*, volume 2238 of *Lecture Notes in Computer Science*, pages 245–258. Springer.

[57] Icking, C., Kamphans, T., Klein, R., and Langetepe, E. (2002). On the competitive complexity of navigation tasks. In *Sensor Based Intelligent Robots*, pages 245–258. Springer.

[58] Icking, C., Kamphans, T., Klein, R., and Langetepe, E. (2005). Exploring simple grid polygons. In Wang, L., editor, *Computing and Combinatorics, 11th Annual International Conference, COCOON 2005, Kunming, China, August 16-29, 2005, Proceedings*, volume 3595 of *Lecture Notes in Computer Science*, pages 524–533. Springer.

[59] Icking, C. and Klein, R. (1995). Searching for the kernel of a polygon - A competitive strategy. In Snoeyink, J., editor, *Proceedings of the Eleventh Annual Symposium on Computational Geometry, Vancouver, B.C., Canada, June 5-12, 1995*, pages 258–266. ACM.

[60] Icking, C., Klein, R., Langetepe, E., Schuierer, S., and Semrau, I. (2004). An optimal competitive strategy for walking in streets. *SIAM Journal on Computing*, 33(2):462–486.

[61] Isbell, J. (1957). An optimal search pattern. *Naval Research Logistics Quarterly*, 4(4):357–359.

[62] Kahn, J. and Saks, M. (1984). Balancing poset extensions. *Order*, 1(2):113–126.

[63] Kim, S., Klein, R., Kübel, D., Langetepe, E., and Schwarzwald, B. (2019). Geometric firefighting in the half-plane. In *Algorithms and Data Structures - 16th International Symposium, WADS 2019, Edmonton, AB, Canada, August 5-7, 2019, Proceedings*, pages 481–494.

[64] Kirkpatrick, D. (2009). Hyperbolic dovetailing. In *European Symposium on Algorithms*, pages 516–527. Springer.

[65] Klein, R. (2006). *Algorithmische Geometrie: Grundlagen, Methoden, Anwendungen*. Springer.

[66] Klein, R., Kübel, D., Langetepe, E., Sack, J., and Schwarzwald, B. (2019a). A new model in firefighting theory. *CoRR*, abs/1911.10341.

[67] Klein, R., Kübel, D., Langetepe, E., Sack, J., and Schwarzwald, B. (2020). A new model in firefighting theory. In Changat, M. and Das, S., editors, *Algorithms and Discrete Applied Mathematics - 6th International Conference, CALDAM 2020, Hyderabad, India, February 13-15, 2020, Proceedings*, volume 12016 of *Lecture Notes in Computer Science*, pages 371–383. Springer.

[68] Klein, R., Langetepe, E., Schwarzwald, B., Levcopoulos, C., and Lingas, A. (2019b). On a fire fighter's problem. *International Journal of Foundations of Computer Science*, 30(2):231–246.

[69] Klein, R., Levcopoulos, C., and Lingas, A. (2018). Approximation algorithms for the geometric firefighter and budget fence problems. *Algorithms*, 11(4):45.

[70] Klötzler, R. (1986). Universale rettungskurven i. *Zeitschrift für Analysis und ihre Anwendungen*, 5(1):27–38.

[71] Klötzler, R. and Pickenhain, S. (1987). Universale rettungskurven ii. *Zeitschrift für Analysis und ihre Anwendungen*, 6(4):363–369.

[72] Koutsoupias, E., Papadimitriou, C., and Yannakakis, M. (1996). Searching a fixed graph. In *International Colloquium on Automata, Languages, and Programming*, pages 280–289. Springer.

[73] Langetepe, E. (2010). On the optimality of spiral search. In Charikar, M., editor, *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 1–12. SIAM.

[74] Langetepe, E. and Kübel, D. (2016a). Optimal online escape path against a certificate. In *15th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2016, June 22-24, 2016, Reykjavik, Iceland*, pages 19:1–19:14.

[75] Langetepe, E. and Kübel, D. (2016b). Optimal online escape path against a certificate. *CoRR*, abs/1604.05972.

[76] Lee, D. and Preparata, F. P. (1984). Euclidean shortest paths in the presence of rectilinear barriers. *Networks*, 14(3):393–410.

[77] Matoušek, J. (2002). *Lectures on discrete geometry*, volume 108. Springer.

[78] McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133.

[79] Minsky, M. L. (1961). Recursive unsolvability of post's problem of "tag" and other topics in theory of turing machines. *Annals of Mathematics*, 74(3):437–455.

[80] Mitchell, J. S. B. (2000). Geometric shortest paths and network optimization. In Sack, J. and Urrutia, J., editors, *Handbook of Computational Geometry*, pages 633–701. North Holland / Elsevier.

[81] Moser, W. O. J. (1991). Problems, problems, problems. *Discrete Applied Mathematics*, 31(2):201–225.

[82] Müller, B. and Reinhardt, J. (1990). *Neural Networks: An Introduction*. Springer Science & Business Media.

[83] Nishizeki, T. and Chiba, N. (1988). *Planar graphs: Theory and algorithms*. Elsevier.

[84] Ogilvy, C. S. (1962). *Tomorrow's math: unsolved problems for the amateur*. Oxford Univ. Press.

[85] Pastor, E., Zárate, L., Planas, E., and Arnaldos, J. (2003). Mathematical models and calculation systems for the study of wildland fire behaviour. *Progress in Energy and Combustion Science*, 29(2):139–153.

[86] Poole, G. and Gerriets, J. (1973). Minimum covers for arcs of constant length. *Bulletin of the American Mathematical Society*, 79(2):462–463.

[87] Preparata, F. P. and Shamos, M. I. (1985). *Computational Geometry - An Introduction*. Texts and Monographs in Computer Science. Springer.

[88] Rao, N. S., Kareti, S., Shi, W., and Iyengar, S. S. (1993). Robot navigation in unknown terrains: Introductory survey of non-heuristic algorithms. Technical report, Oak Ridge National Lab., TN (United States).

[89] Sadakane, K., Takki-Chebihi, N., and Tokuyama, T. (2005). Combinatorics and algorithms for low-discrepancy roundings of a real sequence. *Theoretical Computer Science*, 331(1):23–36.

[90] Schwarzwald, B. (2020). *On Discrete and Geometric Firefighting*. PhD thesis, Rheinische Friedrich-Wilhelms-Universität Bonn.

[91] Shamos, M. I. (1978). *Computational geometry*. PhD thesis, Yale University.

[92] Sharir, M. and Agarwal, P. K. (1995). *Davenport-Schinzel sequences and their geometric applications*. Cambridge University Press.

[93] Sidon, S. (1932). Ein satz über trigonometrische polynome und seine anwendung in der theorie der fourier-reihen. *Mathematische Annalen*, 106(1):536–539.

[94] Skiena, S. S. (1992). Interactive reconstruction via geometric probing. *Proceedings of the IEEE*, 80(9):1364–1383.

[95] Sleator, D. D. and Tarjan, R. E. (1985). Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208.

[96] Storandt, S. (2018). Sensible edge weight rounding for realistic path planning. In *Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, SIGSPATIAL 2018, Seattle, WA, USA, November 06-09, 2018*, pages 89–98.

[97] Takki-Chebihi, N. and Tokuyama, T. (2003). Enumerating global roundings of an outerplanar graph. In *Algorithms and Computation, 14th International Symposium, ISAAC 2003, Kyoto, Japan, December 15-17, 2003, Proceedings, LNCS 2906*, pages 425–433.

[98] Thorup, M. (1999). Undirected single-source shortest paths with positive integer weights in linear time. *Journal of the ACM*, 46(3):362–394.

[99] Thorup, M. (2004). Integer priority queues with decrease key in constant time and the single source shortest paths problem. *Journal of Computer and System Sciences*, 69(3):330–353.

[100] van Leeuwen, J., editor (1990). *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity*. Elsevier and MIT Press.

[101] Weiskircher, R. (1999). Drawing planar graphs. In Kaufmann, M. and Wagner, D., editors, *Drawing Graphs, Methods and Models (the book grow out of a Dagstuhl Seminar, April 1999)*, volume 2025 of *Lecture Notes in Computer Science*, pages 23–45. Springer.

[102] Williams, R. R. (2018). Faster all-pairs shortest paths via circuit complexity. *SIAM Journal on Computing*, 47(5):1965–1985.

[103] Williams, S. W. (2002). Million-buck problems. *The Mathematical Intelligencer*, 24(3):17–20.

[104] Wolfram, S. (1983). Statistical mechanics of cellular automata. *Reviews of modern physics*, 55(3):601.

[105] Wolfram, S. (2002). *A new kind of science*. Wolfram-Media.

[106] Zalgaller, V. A. (2005). A question of bellman. *Journal of Mathematical Sciences*, 131(1):5286–5306.

List of Algorithms

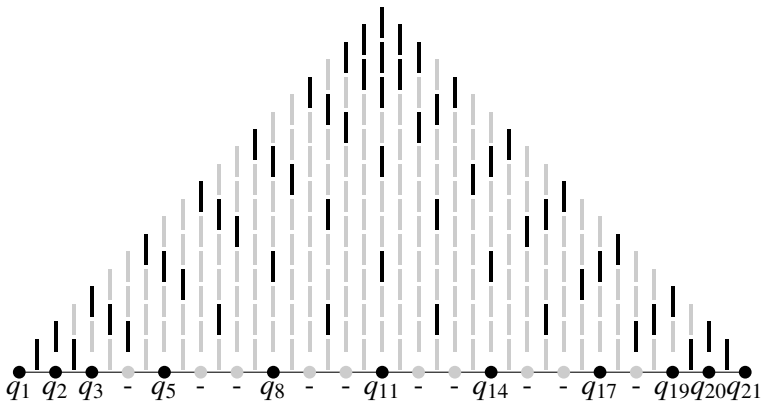
List of Figures

List of Tables

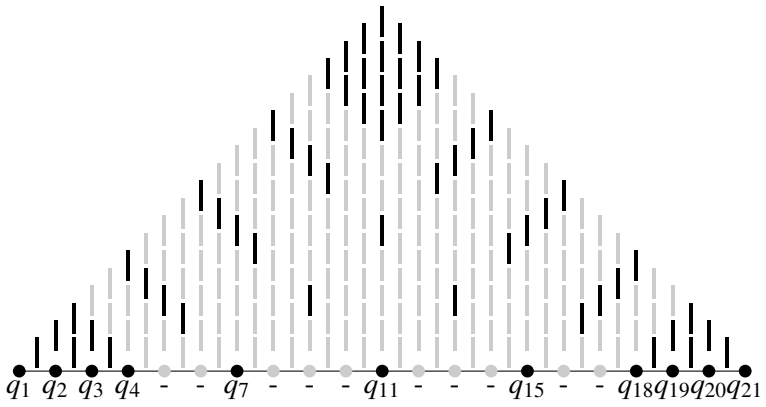
Appendix A

Additional Proofs, Figures, and Pseudocode for Chapter 2

A.1 Removing Query Points from EQUIDIST(21)



(i) Keeping three query points from the left an right, it is possible to leave regular gaps of size two.



(ii) Keeping four query points from the left an right, it is possible to leave regular gaps of size three.

Figure A.1: Removing 10 query points from $\mathcal{Q}_{21}$ without loss of accuracy.

A.2 Proof of Observation 3

Observation 2. *The accuracy of an optimal incremental strategy using $n = 4$ query points is 14.*

Proof. Suppose that an incremental strategy achieves accuracy $a > 14$; thus, we may apply it to the unit interval to locate the target t in a subinterval of size at most $1/a$.

W. l. o. g. we may assume that the first two query points $q_1 < q_2$ are placed symmetrically around $1/2$;[1] thus, we have $q_1 < 1/2$ and $q_2 = 1 - q_1 > 1/2$. Further, we may assume that $t \in [0, 1/2]$, the other case can be proven symmetrically.

Observe that we have to place q_3 such that the accuracy is increased in the third step, i. e., $q_3 < q_2$: Otherwise, q_3 creates no bisector within the interval and the accuracy remains two; this limits the overall accuracy to at most 8 which gives a contradiction.[2] Thus, by placing q_3 we obtain two new bisectors $\mathrm{bs}(q_1, q_3) < \mathrm{bs}(q_2, q_3)$ which subdivide the interval into at least two subintervals (i) $[0, \mathrm{bs}(q_1, q_3)]$ and (ii) $[\mathrm{bs}(q_1, q_3), \min(1/2, \mathrm{bs}(q_2, q_3))]$.

The idea of the proof is as follows. We prove that the sizes of interval (i) and (ii) are at most $3/a$ each; next, we show that q_3 and q_1, as well as q_1 and q_2 lie more than $2/a$ apart; finally, we use these facts to conclude that interval (i) has size larger than $2/a$ which leads to a contradiction. Consider the following chain of facts, where latter build on previous ones.

(A) The size of interval (i) is at most $3/a$, i. e., $\mathrm{bs}(q_1, q_3) \leq 3/a$. Assume to the contrary that the interval was larger; thus, more than two bisectors would be necessary in the fourth step to divide the interval into parts of size at most $1/a$. However, q_4 creates only three bisectors where (at least) one of which does not lie in the interval: $\mathrm{bs}(q_2, q_4) \geq 1/2 \cdot q_2 > 1/4 > 3/a$. Thus, accuracy a could not be achieved on interval (i), which gives a contradiction.

(B) The interval (ii) is bounded by $[\mathrm{bs}(q_1, q_3), \mathrm{bs}(q_2, q_3)]$, i. e., $\mathrm{bs}(q_2, q_3) < 1/2$ and $q_3 < q_1$. If this were not the case, its size would be $1/2 - \mathrm{bs}(q_1, q_3) > 7/a - 3/a = 4/a$ due to Fact (A); thus, more than three bisectors would be necessary in the fourth step to divide the interval into parts of size at most $1/a$. As q_4 creates only three bisectors, accuracy a could not be achieved on this interval, which gives a contradiction.

(C) The distance between q_1 and q_2 is greater than $2/a$. This follows using Fact (B) and (A): $q_2 - q_1 = 1 - 2 \cdot q_1 \geq 1 - 2 \cdot 2 \cdot \mathrm{bs}(q_1, q_3) > 14/a - 4 \cdot 3/a = 2/a$. This implies that, regardless of the position of q_4, the bisectors $\mathrm{bs}(q_1, q_4) < \mathrm{bs}(q_2, q_4)$ are more than $1/a$ apart; moreover, due to Fact (B), it is $\mathrm{bs}(q_3, q_4) < \mathrm{bs}(q_1, q_4)$.

(D) The size of interval (ii) is at most $3/a$, which implies $\mathrm{bs}(q_2, q_3) \leq 6/a$ by Fact (A). Assume to the contrary that the interval was larger; thus, all three bisectors created in the fourth step would be required to divide the interval into parts of size at most $1/a$. However, by Fact (C), the part created by $\mathrm{bs}(q_1, q_4)$ and $\mathrm{bs}(q_2, q_4)$ is of size greater than $1/a$, which gives a contradiction.

[1] Given an optimal query strategy, the left and right subtree of its corresponding query tree are both optimal strategies. Assume to the contrary that q_1, q_2 split the initial interval in two parts of different size; w. l. o. g. we assume that the left part is larger. In both parts, the size of the largest, final subinterval must be of the same size; otherwise, the strategy can be improved. Thus, however, the optimal strategy of the left subtree achieves the same accuracy on a larger interval and can be applied also to the right part to improve the accuracy on this part. This contradicts the assumption that the right subtree's strategy was already optimal and can be used to improve the overall strategy which gives a contradiction.

[2] Placing q_4 creates at most three new bisectors which can split an interval into at most four pars of equal size; hence, the accuracy gain in the fourth step is limited to a factor of four.

(E) The distance between q_1 and q_3 is greater than $2/a$. This can be shown using Fact (D): $q_1 - q_3 = (1-q_2) - q_3 = 1 - 2 \cdot \mathrm{bs}(q_2, q_3) > 14/a - 2 \cdot 6/a = 2/a$.

(F) The distance between q_1 and q_2 is at most $6/a$. Due to Fact (B), it is bounded by bisectors which q_3 creates with q_1 and q_2; due to Fact (D), the size of interval (ii) is at most $3/a$. Thus, the distance between q_1 and q_2 cannot be larger than $6/a$.

(G) The size of interval (i) is greater than $2/a$. From (F) we get $2 \cdot q_1 = q_1 + (1-q_2) = 1 - (q_2 - q_1) > 14/a - 6/a = 8/a$ which we use to bound the size of (i) from below: $\mathrm{bs}(q_1, q_3) - 0 > 1/2 \cdot q_1 = 1/4 \cdot (q_1 + (1-q_2)) = 1/4 - 1/4 \cdot (q_2 - q_1) \geq 1/4 \cdot (1 - 6/a) > 1/4 \cdot (14/a - 6/a) = 2/a$.

Since the size of interval (i) is greater than $2/a$ by Fact (G), at least two bisectors created by q_4 have to split the interval into smaller parts; however, since q_3, q_1 and q_2 lie too far apart, these two bisectors produce a part which is has size greater than $1/a$ which gives a contradiction. □

A.3 Pseudocode for QUERYLEFTSUBINTERVAL

Algorithm 8: QUERYLEFTSUBINTERVAL$([l,r], q_{in}, q_{out}, k, C)$

Input: The interval $[l,r]$ in which t is known to lie; two query points $l \leq q_{in} < r < q_{out}$ such that $r = \mathrm{bs}(q_{in}, q_{out})$; a number $k \geq 2$ of query points to be placed; a scaling factor C.

Output: The routine places a query points q to split the search interval in three parts; then, the search continues recursively depending on in which part t is found to lie.

$q \leftarrow q_{in} - 2 \cdot h(k-1) \cdot C$ ▷ `place query point` q
$\mathrm{bs}_l \leftarrow (q + q_{in})/2, \quad \mathrm{bs}_r \leftarrow (q + q_{out})/2$

if $t \in [l, \mathrm{bs}_l]$ **then**
  QUERYLEFTSUBINTERVAL$([l, \mathrm{bs}_l], q, q_{in}, k-1, C)$
if $t \in [\mathrm{bs}_l, \mathrm{bs}_r]$ **then**
  $\mathrm{bs}'_r \leftarrow \mathrm{bs}_l + (\mathrm{bs}_l - l)$ ▷ `artifical bisector s.th.` $\mathrm{bs}_r \leq \mathrm{bs}'_r$
  QUERYRIGHTSUBINTERVAL$([\mathrm{bs}_l, \mathrm{bs}'_r], q_{in}, q, k-1, C)$
else ▷ $t \in [l, \mathrm{bs}_l]$
  QUERYINTERVAL$([\mathrm{bs}_r, r], k-1)$

Appendix B

Additional Proofs for Chapter 3

B.1 Preconditions of Gal's Theorem

Recall the definition of the sequence of functionals used in Theorem 8:

$$F_l^j(X) \coloneqq \frac{\sum_{i=-l}^{k-1} d\,(x_i, x_{i+1})}{x_{k+1-j}(1 + j \cdot \frac{2\pi}{n})},$$

where $d : \mathbb{R}^2 \to \mathbb{R}$ with $d(x,y) \mapsto \sqrt{x^2 - 2xy\cos\left(\frac{2\pi}{n}\right) + y^2}$, $0 \leq j \leq n$ is fixed and $l \geq k-1$.

Since $d\,(x,y) \geq 0$ for all $x, y \in \mathbb{R}^+$, $F_l^j(X)$ satisfies the *monotonicity condition* [3, (7.20), p. 113]

$$F_{l+1}^j(X) = \frac{\sum_{i=-l-1}^{k-1} d\,(x_i, x_{i+1})}{x_{k+1-j}(1 + j \cdot \frac{2\pi}{n})} \geq \frac{\sum_{i=-l}^{k-1} d\,(x_i, x_{i+1})}{x_{k+1-j}(1 + j \cdot \frac{2\pi}{n})} = F_l^j(X).$$

Moreover, the functionals satisfy the five conditions [3, (7.5)-(7.9), p. 109 f.]; in detail, these conditions are: (7.5) $F_l^j(X)$ is *continuous* since X is a positive sequence; (7.6) $F_l^j(X)$ satisfies the *homogeneity condition*, i. e., for every $\lambda > 0$, it is

$$F_l^j(\lambda X) = \frac{\sum_{i=-l}^{k-1} \sqrt{(\lambda x_i)^2 - 2\cos\left(\frac{2\pi}{n}\right)(\lambda x_i)(\lambda x_{i+1}) + (\lambda x_{i+1})^2}}{\lambda x_{k+1-j}(1 + j \cdot \frac{2\pi}{n})} = F_l^j(X);$$

(7.7) F_l^j satisfies the *unimodality* condition $F_l^j(X+Y) \leq \max\{F_l^j(X), F_l^j(Y)\}$ for every two positive sequences X, Y, since

$$\begin{aligned} F_l^j(X+Y) &= \frac{\sum_{i=-l}^{k-1} d\,(x_i + y_i, x_{i+1} + y_{i+1})}{(x_{k+1-j} + y_{k+-j})(1 + j \cdot \frac{2\pi}{n})} \overset{\text{Le. } 25}{\leq} \frac{\sum_{i=-l}^{k-1} d\,(x_i, x_{i+1}) + d\,(y_i, y_{i+1})}{(x_{k+1-j} + y_{k+1-j})(1 + j \cdot \frac{2\pi}{n})} \\ &= \frac{\sum_{i=-l}^{k-1} d\,(x_i, x_{i+1})}{(x_{k+1-j} + y_{k+1-j})(1 + j \cdot \frac{2\pi}{n})} + \frac{\sum_{i=-l}^{k-1} d\,(y_i, y_{i+1})}{(x_{k+1-j} + y_{k+1-j})(1 + j \cdot \frac{2\pi}{n})} \leq F_l^j(X) + F_l^j(Y). \end{aligned}$$

Finally, condition (7.8) and (7.9) are naturally satisfied.

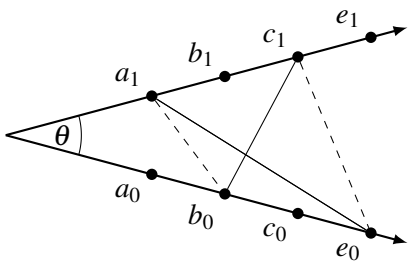


Figure B.1: Positive reals $a \leq b \leq c \leq e$ are mapped on two rays. By triangle inequality, the length of the dashed segments is at most the length of the solid ones.

B.2 Technical Lemmata

Lemma 24. *Let $\theta \in \mathbb{R}$ and $d : \mathbb{R}^2 \to \mathbb{R}$ with $d(x,y) \mapsto \sqrt{x^2 - 2xy\cos\theta + y^2}$. For all $a,b,c,e \in \mathbb{R}_{\geq 0}$ such that $a \leq b \leq c \leq e$, the following inequalities hold:*

1. $d(a,b) \leq d(a,c)$ *and*
2. $d(a,b) + d(c,e) \leq d(a,e) + d(b,c) \overset{1.}{\leq} d(a,e) + d(b,e)$.

Proof. We begin with proving the first inequality. Since $a \leq b \leq c$, there exist $\varepsilon > 0$ and $\delta > 0$ such that $b = a + \varepsilon$ and $c = b + \delta$; moreover, since $d(a,b) \geq 0$, we obtain the equivalence

$$d(a,b) \leq d(a,c) \quad \Leftrightarrow \quad 2a\cos(\theta) \leq 2a + 2\varepsilon + \delta,$$

where the inequality on the right holds $\forall\theta \in \mathbb{R}$ since a, ε, δ are non-negative and $\cos(\theta) \leq 1$.

To prove the second inequality, we map a to two polar coordinates $a_0 \coloneqq (a,0)$, $a_1 \coloneqq (a,\theta)$ on two rays at an angle θ apart; similarly, we proceed for b, c and e, see Figure B.1. Thus, the value of $d(a,b)$ equals the Euclidean distance between corresponding coordinates on different rays; i. e., the distance from a_0 to b_1, which is identical to the distance from a_1 to b_0. By triangle inequality of the Euclidean metric we conclude

$$d(a,b) + d(c,e) = d(a_1,b_0) + d(c_1,e_0) \leq d(a_1,e_0) + d(b_0,c_1) = d(a,e) + d(b,c).$$

□

Lemma 25. *Let $\theta \in \mathbb{R}$ and $d : \mathbb{R}^2 \to \mathbb{R}$ with $d(x,y) \mapsto \sqrt{x^2 - 2xy\cos\theta + y^2}$. For all $x,y,x',y' \in \mathbb{R}_{\geq 0}$ the following inequalities holds: $d(x+y,x'+y') \leq d(x,x') + d(y,y')$.*

For a proof of this lemma we refer to [3, pp. 155 f.].

Appendix C

Proof Sketches for the Lemmata of Chapter 5

Lemma 19 (crossing gadget). *Let* $\mathbf{X}^{\text{in}}$ *be an input-cell and* $\mathbf{X}^{\text{out}}$ *an output-cell of a crossing gadget connected with a straight vertical or horizontal wire.*

If $\mathbf{X}^{\text{in}}$ *is the first cell of the adjacent wire that ignites at time t, the following holds:* $\mathbf{X}^{\text{out}}$ *ignites at time* $t' \leq t+11$*; the state of all cells that do not belong to the wire remains unchanged; and, at time* $t'+1$*, none of the gadget's cells are burning.*

Proof sketch. Set ${}_{l}\mathbf{X}^{\text{in}}$ on fire at time $t=0$, i. e., setting $x({}_{l}\mathbf{X}^{\text{in}},0)=0$. Then, apply the rules of the fire expansion (Definition 12) several times to see ${}_{r}\mathbf{X}^{\text{out}}$ ignites at time $t=11$, i. e., $x({}_{r}\mathbf{X}^{\text{out}},11)=0$. Moreover, at time $t=12$, none of the gadget's cells is burning. By symmetry, the same hold for all other pair of input-output cells.

Note that the choice of the weights for cells where a horizontal wire in $\mathscr{L}_1$ is adjacent to a vertical wire in $\mathscr{L}_2$ is crucial: Since the x-value is two and the y-value is one, it is impossible for such a cell to ignite its neighbour on the other wire. Hence, the state of all cells that do not belong to the wire remains unchanged. This ensures also that a fire can pass along a different wire of the gadget at a moment later in time. □

Lemma 20 (instruction gadget). *Let* $\mathbf{I}^{\text{in}}$ *be an input-cell,* $\mathbf{I}^{\text{out}}$ *be an output-cell, and* $\mathbf{I}^{\text{cen}}$ *be the centre-cell in* $\mathscr{L}_1$ *of the basic instruction gadget. If* $\mathbf{I}^{\text{in}}$ *ignites at time t,* $\mathbf{I}^{\text{cen}}$ *ignites at time* $t' \leq t+8$*; if* $\mathbf{I}^{\text{cen}}$ *ignites at time t,* $\mathbf{I}^{\text{out}}$ *ignites at time* $t'' \leq t+8$*. The state of all other input-cells, except for* $\mathbf{I}^{\text{in}}$*'s direct neighbour, remains unchanged; and, at time* $t''+1$*, none of the gadget's cells are burning.*

Proof sketch. If $\mathbf{I}^{\text{in}}$ is the first cell of the gadget that ever ignites, the lemma can be verified by setting $x(\mathbf{I}^{\text{in}},0)=0$ and applying the rules of the fire expansion (Definition 12). Note that none of the other input-cells can ignite due to the choice of weight of their inner neighbour. The fire burns for a maximum number of 16 steps when $\mathbf{I}^{\text{in}}$, $\mathbf{I}^{\text{out}}$ lie in $\mathscr{L}_2$ of the gadget.

If $\mathbf{I}^{\text{in}}$ is not the first cell of the gadget that ever ignites, then $\mathbf{I}^{\text{out}}$ already ignited a time earlier than $t+16$ which does not invalidate the lemma. □

Lemma 21 (program gadget). *Let* $\mathbf{P}_i^{\text{in}}$ *be an input-cell and* $\mathbf{P}_i^{\text{out}}$ *be an output-cell of the program gadget which are connected to the* I_i*-instruction gadget with a horizontal or vertical wire. Moreover, let* $\mathbf{I}_i^{\text{cen}}$ *be the centre-cell of the* I_i*-instruction gadget,* $T \coloneqq 12(n-1)+8$*, and n the number of instructions of 2M.*

If $\mathbf{P}_i^{\text{in}}$ ignites at time t, then $\mathbf{I}_i^{\text{cen}}$ ignites at time $t' \leq t+T$. Moreover, if $\mathbf{I}_i^{\text{cen}}$ ignites at time t, then ${}_r\mathbf{P}_i^{\text{out}}$ (${}_l\mathbf{P}_i^{\text{out}}$) ignites at time $t'' \leq t+T$ if and only if I_i is an addition (subtraction) instruction for register r_1; and, ${}_t\mathbf{P}_i^{\text{out}}$ (${}_b\mathbf{P}_i^{\text{out}}$) ignites at time $t'' \leq t+T$ if and only if I_i is an addition (subtraction) instruction for register r_2. The state of all other boundary cells remains unchanged and, at time $t''+1$, none of the gadget's cells are burning.

Proof sketch. Let $\mathbf{I}_i^{\text{in}}$ be the input-cell of the I_i-instruction gadget to which $\mathbf{P}_i^{\text{in}}$ is connected to via the input-wire. All cells of this input-wire are alive since fire cannot expand along input-wire into the reverse direction, fire does not ignite input-cells of the instruction gadget from inside, fire does not leap over from adjacent wires, and no cells ignite spontaneously. The input-wire passes through at most $n-1$ crossing gadgets, each of which requires twelve steps by Lemma 19. Hence, by an inductive argument, $\mathbf{I}_i^{\text{in}}$ ignites at time $t+12(n-1)$ the latest. By Lemma 20, $\mathbf{I}_i^{\text{cen}}$ ignites at most eight steps later than that.

Similarly, one can prove the second implication. Note that, by construction, $\mathbf{I}_i^{\text{out}}$ points into the correct direction depending on the type of the instruction of I_i. Moreover, the fire expands only along a single wire at a time and does not change the state (not even the x- or y-value) of any other boundary cell. □

Lemma 22 (connector gadget). *In a basic connector gadget, let ${}_t\mathbf{C}_j^{\text{in}}, {}_b\mathbf{C}_j^{\text{out}}$ be the input-cell and output-cell of the $(2j)$-th vertical wire from the left. Moreover, let ${}_t\mathbf{C}_i^{\text{out}}$ be the output-cell of the $(2i-1)$-th vertical wire from the left and let c be a cell of the attached wire that runs through both layers.*

If ${}_t\mathbf{C}_j^{\text{in}}$ ignites at time t, then ${}_b\mathbf{C}_j^{\text{out}}$ ignites at time $t' \leq t+12n$. Moreover, if c ignites at time t, then ${}_t\mathbf{C}_i^{\text{out}}$ ignites at time $t' \leq 60n$. In both cases, the state of all other boundary cells remains unchanged and, at times $t'+1$ and $t''+1$, none of the gadget's cells are burning.

Proof sketch. Since the wire from ${}_t\mathbf{C}_j^{\text{in}}$ to ${}_b\mathbf{C}_j^{\text{out}}$ runs through at most n crossing gadgets, the first implication follows from applying Lemma 19 repeatedly.

For the second implication, the wire from c to ${}_t\mathbf{C}_i^{\text{out}}$ runs through at most $2n$ crossing gadgets to the right, at most n crossing gadgets upwards, and at most $2n$ crossing gadgets back to the left. Altogether, the wire passes through at most $5n$ crossing gadgets each of which requires twelve steps. The implication follows by applying Lemma 19 repeatedly and taking the placement of corners into account. □

Lemma 23. *Let ${}^{(b_1,b_2)}\mathbf{I}_i^{\text{cen}}$ be the centre-cell of the I_i-instruction gadget within the box gadget of coordinates (b_1,b_2) where $I_i \neq I_n$.*

If ${}^{(b_1,b_2)}\mathbf{I}_i^{\text{cen}}$ ignites at time t, exactly one of the following implications holds:

- ${}^{(b_1+1,b_2)}\mathbf{I}_j^{\text{cen}}$ *ignites at time t' if $I_i = (i,r_1,j)$;*
- ${}^{(b_1,b_2+1)}\mathbf{I}_j^{\text{cen}}$ *ignites at time t' if $I_i = (i,r_2,j)$;*
- ${}^{(b_1-1,b_2)}\mathbf{I}_k^{\text{cen}}$ *ignites at time t' if $I_i = (i,r_1,j,k)$ and $b_1 > 0$;*
- ${}^{(b_1,b_2-1)}\mathbf{I}_k^{\text{cen}}$ *ignites at time t' if $I_i = (i,r_2,j,k)$ and $b_2 > 0$;*
- ${}^{(b_1,b_2)}\mathbf{I}_j^{\text{cen}}$ *ignites at time t' if $I_i = (i,r_1,j,k)$ and $b_1 = 0$;*
- ${}^{(b_1,b_2)}\mathbf{I}_j^{\text{cen}}$ *ignites at time t' if $I_i = (i,r_2,j,k)$ and $b_2 = 0$;*

where t' is bounded by a constant depending on n, the number of instructions of 2M.

Proof sketch. Let ${}^{(b_1,b_2)}\mathbf{I}_i^{\text{cen}}$ ignite at time t.

If instruction I_i is the halting instruction, then the fire extinguishes after at most eight steps due to the construction of the I_n-instruction gadget; note that no boundary cell of the I_i-instruction gadget changed its state or values.

If $I_i = (i, r_1, j)$, then ${}^{(b_1,b_2)}_{r}\mathbf{P}_i^{\text{out}}$ ignites at time $t' \leq t + T$, by Lemma 21. Since I_i is an addition instruction, no corner has been placed along the wire from ${}^{(b_1,b_2)}_{r}\mathbf{P}_i^{\text{out}}$ to ${}^{(b_1,b_2)}_{r}\mathbf{B}_i^{\text{out}}$ of the l -connector gadget; thus, by Lemma 22, ${}^{(b_1,b_2)}_{r}\mathbf{B}_i^{\text{out}}$ ignites at time $t' \leq t + T + 12n$. Now, ${}^{(b_1,b_2)}_{r}\mathbf{B}_i^{\text{out}}$ is adjacent to ${}^{(b_1+1,b_2)}_{l}\mathbf{B}_i^{\text{in}}$ which ignites in the next step. However, by construction, a corner has been placed to connect ${}^{(b_1+1,b_2)}_{l}\mathbf{B}_i^{\text{in}}$ to the j-th output-wire of the l -connector gadget. Hence, after passing through at most $(n-1)/2$ crossing gadgets, the fire ignites a cell c on the j-th output-wire of the connector gadget after another $12(n-1)/2$ steps. By Lemma 22, ${}^{(b_1+1,b_2)}_{r}\mathbf{C}_j^{\text{out}}$ of the l -connector gadget ignites after at most $60n$ steps; since it is adjacent to ${}^{(b_1+1,b_2)}_{l}\mathbf{P}_j^{\text{in}}$ by construction, this cell ignites in the next step. Finally, by Lemma 21, ${}^{(b_1+1,b_2)}\mathbf{I}_j^{\text{cen}}$ ignites after another T steps at time $t' \leq t + T + 12n + 12(n-1)/2 + 60n + T \leq t + 30(n-1) + 72n$.

Similarly, one can prove the correctness for all other types of instructions and values of (b_1, b_2). □

Index