Doctoral Thesis

Computing routes and trees under uncertainty

Author(s):
Montanari, Sandro

Publication Date:
2015

Permanent Link:
https://doi.org/10.3929/ethz-a-010608560

Rights / License:
In Copyright - Non-Commercial Use Permitted
COMPUTING ROUTES AND TREES UNDER UNCERTAINTY

A thesis submitted to attain the degree of
Doctor of Science of ETH Zurich

Presented by Sandro Montanari
born 27.05.1986
citizen of Italy

Master of Science University of Pisa
Bachelor of Science University of Pisa

Accepted on the recommendation of

Prof. Dr. Peter Widmayer ETH Zurich
examiner

Dr. Matúš Mihalák Maastricht University
co-examiner

Prof. Dr. Anita Schöbel University of Göttingen
co-examiner

2015
Abstract

In this thesis we investigate algorithmic problems related to the computation of shortest paths and minimum spanning trees in graphs with uncertain edge weights. Two target applications motivate and drive our efforts.

The first application is the routing of private vehicles in road networks. To deal with the uncertainty, we adopt a general framework proposed by Buhmann [27]. We consider two different ways to apply this framework to the routing of cars in road networks, resulting in two criteria called maximum similarity and first intersection. We design an exponential-time algorithm to compute optimum paths according to these criteria and empirically demonstrate the good quality of the produced routes on real-world instances. We then turn to the design of efficient algorithms for the first intersection criterion by engineering a bi-directional algorithm with good practical performance. We also investigate speeding-up a particular step of the maximum similarity criterion by considering a technique known as Markov chain Monte Carlo in the context of sampling simple paths in planar graphs.

The second application concerns the computation of shortest paths and spanning trees in graphs where each vertex is associated with a point in the plane. We model the uncertainty according to the imprecise points model, where each point is replaced by an own occurrence region, and the “real” point can be located anywhere in its region. We consider the computation of the smallest/largest value that a geometric object or measure can attain for a given input of occurrence regions. In particular, for the shortest path problem we consider the case where the occurrence regions are axis-aligned rectilinear polygons and distances are measured with the $L_1$ metric. We design for this case an efficient algorithm computing the smallest weight that can be attained by a shortest path
between two given vertices of the graph. We also show that the problem of computing the largest weight of a shortest path between two given vertices is hard to approximate for any approximation factor \((1 - \epsilon)\) with \(\epsilon < 1/4\) for any metric \(L_p, p \geq 1\). We then turn at the computation of a spanning tree with minimum \textit{weight}, defined as the sum of the lengths of its edges, or \textit{diameter}, defined as the length of its longest simple path. We show that there does not exist an FPTAS to compute the smallest weight that can be attained by a spanning tree for imprecise points in axis-aligned segments. We also provide efficient algorithms computing the smallest diameter of a spanning tree for imprecise points in disks. The run-time of these algorithms ranges from \(O(n^5)\) in the case where the \(n\) disks are unit and disjoint to \(O(n^9)\) in the case where the disks have arbitrary radii and are possibly intersecting.
Sommmario

In questa tesi studiamo problemi algoritmici connessi al calcolo di cammini e alberi di copertura minimi in grafi in cui il peso di ciascun arco è incerto. Due applicazioni di riferimento sono alla base e guidano i nostri sforzi.

La prima applicazione riguarda la commutazione per veicoli privati in reti stradali. Per gestire l’incertezza usiamo un metodo generale proposto da Buhmann [27]. Consideriamo due diversi modi per applicare questo metodo alla commutazione per veicoli privati in reti stradali ottendendo in questo modo due criteri, chiamati *maximum similarity* e *first intersection*. Sviluppiamo un algoritmo per calcolare cammini ottimali secondo questi criteri in tempo esponenziale e dimostriamo empiricamente la buona qualità dei cammini calcolati su istanze prese dal mondo reale. Ci volgiamo quindi alla realizzazione di algoritmi efficienti per il criterio first intersection sviluppando un algoritmo bidirezionale con buone prestazioni in pratica. Studiamo anche come accelerare uno stadio specifico del criterio maximum similarity considerando una tecnica nota con il nome di *Markov chain Monte Carlo* nel contesto del campionamento di cammini semplici in grafi planari.

La seconda applicazione riguarda il calcolo di cammini e alberi di copertura minimi in grafi dove ogni vertice è associato con un punto nel piano. Descriviamo l’incertezza usando il modello *punti imprecisi*, in cui ogni punto viene sostituito con una *regione di incertezza* e il punto “reale” si può trovare ovunque nella sua regione. In particolare, per il problema dei cammini minimi consideriamo il caso in cui le regioni di occorrenza sono allineate agli assi e rettilinee e le distanze sono misurate secondo la metrica $L_1$. Per questo caso sviluppiamo un algoritmo efficiente per calcolare il più piccolo peso che può essere ottenuto da un cammino minimo tra due vertici del grafo. Dimostriamo anche che il problema
di calcolare il peso più grande che può essere ottenuto da un cammino minimo tra due vertici è difficile da approssimare per ogni fattore di approssimazione \((1 - \epsilon)\) con \(\epsilon < 1/4\). Ci volgiamo quindi al calcolo di alberi di copertura con minimo peso, definito come la somma delle lunghezze dei suoi archi, o minimo diametro, definito come la lunghezza del suo cammino semplice più lungo. Dimostriamo che non può esistere una FPTAS per calcolare il più piccolo peso che può essere ottenuto da un albero di copertura per punti imprecisi in segmenti allineati alle assi. Proponiamo anche algoritmi efficienti per calcolare il più piccolo diametro di un albero di copertura per punti imprecisi in dischi. Il tempo di esecuzione di questi algoritmi varia tra \(O(n^5)\) nel caso in cui gli \(n\) dischi sono unitari e disgiunti e \(O(n^9)\) nel caso in cui i dischi hanno raggio arbitrario e possono intersecare.
First of all I would like to thank Peter Widmayer, without whom I would never have been here in the first place. He has given me the opportunity to look upon the world of scientific research from a very comfortable and welcoming environment and was always available for helpful discussions and support.

Second I thank my co-examiners Matúš Mihalák and Anita Schöbel, who took time away from their always full and always busy schedules to read my thesis.

Third come my co-authors Yann Disser, Chih-Hung Liu and Paolo Penna (and some of the above), who helped me make the results of this thesis possible.

Last are my past and present colleagues and fellow PhD students, in particular my closest ones, Katerina Böhmová and Tobias Pröger. May the European projects never torment you again!!!

And Silvia, of course.
Contents

Preface 1

I Robust Route Planning 5

1 Models of Uncertainty 7
   1.1 Adaptive Models ........................................ 8
   1.2 Non-adaptive Models ..................................... 10
   1.3 Recoverable Robustness ................................ 16
   1.4 Stochastic Models ....................................... 17

2 Routing via Approximation Sets 21
   2.1 Approximation Sets Framework .......................... 22
   2.2 Maximum Similarity & First Intersection ............. 23
   2.3 Speed-up of First Intersection ......................... 31

3 Sampling Paths According to Their Length 47
   3.1 A Markov Chain for Planar Graphs .................... 51
   3.2 Analysis of $M_{\text{paths}}$ ............................. 52
   3.3 A Rapidly Mixing Chain ................................. 58
   3.4 A Dynamic Programming Sampler ....................... 64

II Optimization for Imprecise Points 67

4 Models of Imprecision 69
   4.1 Imprecise Points ........................................ 69
   4.2 Alternative Models ...................................... 72

5 Shortest Paths 75
Preface

For a computer scientist, the term *optimization* is always associated with the task of selecting a particular element from a set of possible alternatives. Given his passion for numbers, the computer scientist will assign a “score” to each element in the set, and then design an algorithm to compute the element with the best score. If his motivation is application-driven, the scores will be derived from observations and measurements of the real world. It is well-known however that no measurement system is perfect. The use of an imperfect system may lead to select an element that, while being the best one *on paper*, is a very poor one in reality.

The ambitious goal of *optimization under uncertainty* is to look beyond the poor measurements offered by the measurement systems and compute a best element without being blinded by erroneous numbers. Even though this goal is of course impossible to achieve in its full generality, not all hope is lost. A possible solution to our dilemma might for some cases be found by looking back at the application driving our efforts. Advanced knowledge on the application environment might give us the tools to overcome the uncertainty in the data, for example by enabling us to detect and discard those measurements that are indeed erroneous.

The separation between observation and computation dictates that the efforts invested in the task of optimization under uncertainty are also twofold:

1. A first phase is dedicated to the *modeling* of the uncertainty in the concrete optimization problem on hand. In this phase we look at the rules governing the uncertain data and try to integrate them into a model unambiguously defining how an optimum solution should look like.
2. A second phase concerns the computation of such an optimum solution. It involves the design of algorithms and procedures to identify the desired solution within a suitable degree of accuracy, according to the usual limits and bounds of computational complexity.

In this thesis, we assume an algorithmic perspective on the computation of optimum solutions for uncertain problems related to minimum spanning trees and shortest paths in graphs. Two applications motivate our work.

The first application is the routing of private vehicles in road networks. That is, we look at the computation of fast routes in graphs where the vertices represent locations and the edges represent a network of streets connecting these locations. The uncertainty in this case arises because of factors like traffic congestion, weather conditions, or accidents and influences the travel time required to traverse the streets.

The second application concerns the computation of spanning trees and shortest paths in geometric graphs. Each vertex of the graph is associated with a point in the plane and the edges correspond to connections between these points. The uncertainty in this case arises because we consider locations of points measured with some imprecise system, like the Global Positioning System.

Even though our interests lie mainly in the computational phase, we do not ignore modeling completely. For each target application we first review the plethora of models available in order to settle for the most appropriate one for our purposes. We do not however attempt to design completely new models.

Overview of Results

Part I. In the first part of this thesis we look at the routing of private vehicles in road networks. As a preliminary step and before presenting our own contributions, in Chapter 1 we survey the various models for optimization under uncertainty that in the literature have been applied to the problem of routing. Our purpose is to provide a state-of-the-art picture of the field and highlight the most important results.

In Chapter 2 we focus on a model for optimization under uncertainty proposed by Buhmann et al. [27, 28]. We consider two different ways
of applying this general model to the shortest path problem, resulting in the maximum similarity and first intersection criteria. We design an exponential-time algorithm to compute optimum paths according to these criteria and empirically demonstrate the good quality of the produced routes on real-world instances. We then turn to the design of efficient algorithms for the first intersection criterion by engineering a bi-directional algorithm with good practical performance.

In Chapter 3 we investigate the speed-up of a particular step of the maximum similarity criterion, requiring to sample from the set of all s-t paths of a graph according to a given distribution. We consider the technique of Markov chain Monte Carlo and design a Markov chain that can be used to sample simple s-t paths in an (undirected, unweighted) planar graph. This chain can be tuned so that the probability of sampling a path depends on its length. We show that this chain is always ergodic and thus it converges to the desired sampling distribution for any planar graph. While this chain is not rapidly mixing in general, we prove that a simple restricted variant is. The restricted chain samples paths on a 2D lattice which are monotone in the vertical direction.

**Part II.** In the second part we turn to the optimization of geometric problems under uncertainty according to a model known as imprecise points. In this model we assume that the location of the input points in the geometric problem on hand are not known exactly. For each point, instead of its exact location we are given an occurrence region, and the “real” point can be located anywhere in its region. The question we ask in this setting is the smallest/largest value that a geometric object or measure can attain for a given input of occurrence regions. A brief overview of the literature related to this model is presented in Chapter 4.

In Chapter 5 we apply this model to the shortest path problem. We first consider a minimization variant, in which we want to compute the smallest weight that can be attained by a shortest path between two given imprecise points of a geometric graph. We show that, if the occurrence regions of the imprecise points are axis-aligned rectilinear polygons and distances are measured with the $L_1$ metric, the problem can be solved efficiently. This results contrasts with the general case, that is $\text{NP}$-hard for any metric $L_p$ with $p \geq 1$ even if the regions consist of segments whose angles with the $x$-axis are in $\{0, \pm \pi/4, \pi/2\}$. For the maximization variant, we prove the problem to be hard to approximate for any approximation factor $(1 - \epsilon)$ with $\epsilon < 1/4$, even
if the occurrence regions consist only of vertically aligned segments. In the special case where the underlying geometric graph is a path, we show how to solve the problem efficiently for occurrence regions shaped as simple polygons that are not necessarily convex and possibly intersecting.

In Chapter 6 we apply the imprecise points model to the problem of computing a spanning tree with minimum *weight*, defined as the sum of the lengths of its edges, or *diameter*, defined as the length of its longest simple path. We show that there does not exist a fully-polynomial time approximation scheme (FPTAS) to compute the smallest weight that can be attained by a spanning tree of a given set of imprecise points even if the occurrence regions consist only of axis-aligned segments, answering an open question of Dorrigiv et al. [42]. For the computation of the smallest diameter that can be attained by a spanning tree for imprecise points in disks we provide polynomial time algorithms. The run-time of these algorithms ranges from $O(n^5)$ in the case where the $n$ disks are unit and disjoint to $O(n^9)$ in the case where the disks have arbitrary radii and are possibly intersecting.
Route planning, or simply “routing”, is the task of computing quick paths between two locations of a given network. From an algorithmic viewpoint, this task is typically translated to the shortest path problem on weighted directed graphs. Since in this thesis we focus on issues arising in route planning for private vehicles, the graphs we consider represent road networks. A distinguishing feature of a road network is that the weight of an edge corresponds to the travel time to cross the corresponding road. These weights are usually an aggregation (e.g., average) of measured travel times of many individual cars over many similar time points. Aggregated values provide a good estimate of the expected travel time, yet only over a large amount of past days. They say little about deviations of the actual travel times and about per-day nuances of the traffic situations. In fact, a shortest path computed from such aggregated values can perform substantially bad on one particular day. There exists no universally accepted or standard technique for dealing with such uncertain situations; several different alternatives have been proposed in the literature, but none emerged as the dominant one. In Chapter 1 we review those techniques that have been applied to routing and highlight the main results. We then focus on a general framework for optimization under uncertainty proposed by Buhmann et al. [28]. In Chapter 2 we apply this framework to the routing of private vehicles in road networks and investigate the related algorithmic problems. In Chapter 3 we look further into a particular step of the approach by Buhmann requiring to sample simple paths in graphs according to a given distribution.
Chapter 1

Models of Uncertainty

Existing techniques for dealing with uncertainty assume that the uncertain data is not arbitrary, but follows some rules. We might not have perfect knowledge of such rules, but we can estimate a model describing them, at least up to some degree of accuracy. A vast majority of the models that have been proposed in the literature can be classified into two main categories. Following the somewhat misleading but de facto standard terminology of the field, we thus discuss models that are robust or stochastic. In this chapter we review those models that have been applied to the task of route planning. Since the shortest path problem lies at the core of routing, we focus on this particular problem.

With robust models we do not make any assumption on the probability distribution governing the uncertain data. They are used in those cases where, for example, we do not know “how likely” it is to find a road congested. We then wish to be prepared for every situation that we may encounter, in particular for the most unfavorable one. We consider models of this kind in Sections 1.1 to 1.3.

With stochastic models we assume that we do know how likely it is for a road to be congested. We have available an estimation of the probability distribution governing the uncertain data and rely on this knowledge when looking for a solution. Since the main focus of this thesis is robustness, we provide only a brief overview of the most prominent stochastic models in Section 1.4.

Within the above categories we provide a further classification, based on
when the uncertain data becomes “certain”. When this situation arises we might allow to modify the chosen path if we notice that another one produces a better result. For example, while driving with a car we might be listening to the news on the radio and hear that there is a traffic jam ahead of us. We may then decide to leave the current route to avoid the congestion. We define a model to be adaptive if further information on the uncertainty is gained piece by piece and we are allowed to modify the chosen path to react to this new information. If the uncertain data is given completely after the path has been chosen and the selected path cannot be modified, we denote the model as non-adaptive. There are also models where, once the uncertain data has been revealed, we are allowed some limited actions in order to improve or recover our current solution. We denote these models as recoverable.

1.1 Adaptive Models

When shortest paths are computed under uncertainty, some essential knowledge about the graph is missing or not complete. For adaptive models, pieces of the missing information are acquired after the trip begins and we are allowed to deviate from the chosen path and follow a different one. Given their nature, adaptive models ask for an online algorithm to compute a solution. To evaluate the quality of a solution produced by an online algorithm we use the standard measures of competitive ratio [8, 21] and Max/Max ratio [14, 21]. Both measures consider all possible input sequences of the online algorithm, i.e., the pieces of unknown information in the order in which the algorithm receives them. For shortest path problems, we say that an algorithm has competitive ratio $M$ if the weight of the path it computes for any input sequence is never more than $M$ times the weight of an offline optimum path, i.e., a shortest path computed when the whole uncertain information has been revealed. We further say that the Max/Max ratio of an algorithm is $M$ if the weight of the path it computes for any input sequence of length $K$ is never more than $M$ times the weight of an offline optimum for any input sequence of the same length. Note that, for the purpose of comparing two online algorithms for the same problem, this measure simply corresponds to the comparison of their worst-case input sequences.
Canadian traveler. Papadimitriou and Yannakakis [114] consider a first case of adaptive shortest path problem under the name of Canadian Traveler Problem (CTP). The prototypical example for this problem is a Canadian driver during winter. He knows the best route between two cities, but roads might be blocked by snow and this information is discovered only upon reaching one of the endpoints of the road. The formal goal of the CTP is the same as the classical shortest path problem: given a graph $G = (V, E)$ with weights $w_e \in \mathbb{R}^+$ for every edge $e \in E$, we ask for a shortest path between two vertices $s, t \in V$. The difference is that only an unknown set of edges $E_{\text{free}} \subseteq E$ can be traversed. Whether or not an edge is usable or blocked is discovered when one of its endpoints is reached (see Figure 1.1 for an example). The input sequence of an online algorithm for the CTP is the list of blocked edges. At any step, the blocked edges are revealed for the current vertex and the algorithm outputs the next edge to be followed. To compute the output, the algorithm can only use the underlying graph $G$ given as input at the beginning and the discovered information on $E_{\text{free}}$. An offline optimum is a shortest path in the graph using only edges from $E_{\text{free}}$. The authors propose also a simple generalization of the CTP, where we are given a graph in which every edge has two possible weights. The actual weight is one of the two and this information is discovered when one of the endpoints of $e$ is reached. They show that, both for the CTP and its generalization, the problem of computing a path achieving a given competitive ratio is PSPACE-complete.

Bounded Canadian traveler. Bar-Noy and Schieber [11] consider the CTP in a setting where a bound $k$ on the overall number of edges that can be blocked is known. They show that the problem of computing a path with minimum Max/Max ratio can be solved in polynomial time if $k$ is constant, but it becomes PSPACE-complete if $k$ is part of the
input. Westphal [129] considers the competitive ratio of this problem and shows the lower bounds of $2k + 1$ and $k + 1$ on the ratios of any deterministic and randomized algorithm, respectively. The author also provides a simple deterministic algorithm for undirected graphs matching the lower bound. The algorithm follows a shortest $s$-$t$ path until the first blocked edge is encountered; when this happens, it drives back to $s$ and tries to get to $t$ again by following a shortest path in the updated graph. This procedure is repeated until the vertex $t$ is reached. A randomized algorithm matching the lower bound of $k + 1$ has been shown only for graphs where all $s$-$t$ paths are vertex-disjoint [15], while for general graphs the $2k + 1$ bound has been surpassed only by an $o(1)$ factor [36]. Liao and Huang [86] prove similar bounds and algorithms for the generalization where every edge has two possible weights, and the actual weight of an edge is discovered when one of its endpoints is reached.

**Recoverable edges.** Bar-Noy and Schieber [11] study a further variant of the CTP differing from the original problem in three aspects. The first aspect is that every time a vertex is reached a new list of adjacent blocked edges is given, regardless of the previous input of the algorithm. The second is that the algorithm can reject a list that it has received as input and require a new one by “waiting” a fixed time at the current vertex. The last aspect is that the overall number of edges that can be blocked is bounded by a constant $k$. After $k$ blocked edges have been discovered, it is assured that every list is empty for every vertex, therefore the online algorithm can safely follow a shortest path from the current vertex to the destination. Note that the blocked edges in a rejected list still count towards the bound $k$. For small waiting times, the authors design an algorithm with minimum Max/Max ratio and overall run-time of $O(k^2m + kn \log n)$. Liao and Huang [86] combine the backtracking algorithm by Westphal with a greedy strategy to achieve, for this setting, a competitive ratio of $2k + 1$.

### 1.2 Non-adaptive Models

The difference between adaptive and non-adaptive models is that a chosen path cannot be modified or replaced. We therefore want to guarantee that our decision is robust (in a sense that will be specified
later) with respect to every possible realization of the uncertain information. Without loss of generality, we assume only the edge weights of a graph to be uncertain and not its topology. It is possible to simulate the absence of an edge in the graph by assigning a suitably large cost to it.

At a general level, Kouvelis and Yu [83] show two possible ways to define when a solution of an optimization problem is considered robust: the min-max and the min-max regret criteria. The min-max criterion defines a robust solution in absolute terms, as one whose maximum weight among all possible realizations of the uncertain data is minimized. It represents a conservative viewpoint, where we expect the worst situation to be the one we will actually have to face and want our losses to be minimized in this case. The min-max regret criterion defines a robust solution as one minimizing the maximum difference (or ratio) between its weight and the weight of an optimum one in all realizations. This criterion is preferred when we wish to benchmark our decision a posteriori, in terms of gain losses. When the ratio instead of the difference between the weights of a robust solution and that of an optimum one is considered, the criterion is called the min-max relative regret.

In the remainder of this chapter we assume to be given a graph \( G = (V, E) \) with non-negative weights \( w_e \) for every edge \( e \in E \) and two vertices \( s, t \in V \). We consider the integer linear program formulation of the shortest path problem

\[
\begin{align*}
\text{arg min} & \quad \sum_{e \in E} x_e \cdot w_e \\
\text{subject to} & \quad \forall u \in V : \sum_{e \in E, e = (u,v)} x_e - \sum_{e \in E, e = (v,u)} x_e = \begin{cases} 1 & \text{if } u = s, \\ -1 & \text{if } u = t, \\ 0 & \text{otherwise,} \end{cases} \\
& \quad \forall e \in E : x_e \in \{0,1\}.
\end{align*}
\]

The vector \( x = (x_e) \) yielded by this program corresponds to a shortest \( s-t \) path, if such a path exists. If the edge weights are strictly positive, the shortest path will be simple, i.e., without cycles.

There are many different concrete models for robust optimization. The main difference between them is in how the possible realizations of the uncertain information are described. This description is application-oriented and it affects directly the feasibility of computing a robust path.
In the following, we modify the above linear program to highlight the
difference between the various models and the classical shortest path
problem. Our assumption that only the edge weights are subject to
uncertainty implies that only the objective function of the program is
uncertain and not its constraints.

**Discrete scenarios.** Yu and Yang [131] investigate a model where the
realizations of the uncertain information are given as a predetermined
set of \( N \) edge weight functions. Each realization, in this case also called
scenario, is specified by a value \( r \in \{1, \ldots, N\} \); the weight of \( e \in E \) in
scenario \( r \) is \( w_e^{(r)} \). Figure 1.2 shows an example of this setting. A
robust path according to the min-max criterion minimizes its maximum
weight among all scenarios. We can write this path as

\[
\arg \min_x \max_{r \in S} \sum_{e \in E} x_e \cdot w_e^{(r)}
\]

subject to (1.1) and (1.2).

Similarly, a robust path according to the min-max regret criterion mini-
mizes the maximum difference between its weight and the weight of a
shortest path over all possible scenarios. Such a path can be expressed as

\[
\arg \min_x \max_{r \in S} \left\{ \sum_{e \in E} x_e \cdot w_e^{(r)} - sp_r \right\}
\]

subject to (1.1) and (1.2),

where \( sp_r \) is the weight of a shortest path in scenario \( r \). For the min-max
relative regret criterion, it is sufficient to replace the minus sign in the
objective function with the division sign. It is easy to see that these three
criteria, even though they might appear similar, can return different
paths. This if for example the case for Figure 1.2, where the optimum for the min-max criterion is $e_2$, for the min-max regret criterion is $e_3$, and for the min-max relative regret is $e_1$.

Note that the objectives of the above programs are not linear, making the problem of computing robust paths hard for all three criteria. In particular, Yu and Yang [131] consider computing a robust path according to the min-max and the min-max regret criteria on a special class of graphs called *layered graphs*. In a $k$-layered graph $G = (V, E)$ there is a partition of the vertices $V = V_0 \cup \cdots \cup V_{k+1}$ with $V_0 = \{s\}$ and $V_{k+1} = \{t\}$ in which edges exist only between $V_l$ and $V_{l+1}$, for $l = 0, \ldots, k$. The authors show both problems to be NP-hard even for 2-layered graphs and only 2 scenarios. They also present algorithms for general graphs based on dynamic programming with run-time pseudo-polynomial in the number of scenarios $N$ and prove both problems to be strongly NP-hard if $N$ is part of the input. Aissi et al. [5, 7] look further at the case where $N$ is constant and provide polynomial-time approximation schemes (FPTAS). On the other hand, Kasperski and Zieliński [78] prove that neither of the two problems can be approximated within a constant ratio if $N$ is not constant. An approximation ratio of $N$ can however be obtained by computing the *mid-point solution*, that is a shortest path in the single scenario instance obtained by averaging the weights among all scenarios [7].

We note that it is straightforward to adapt the hardness proofs for the min-max (regret) criterion to the min-max relative regret criterion. We also observe that an optimum path for the min-max relative regret criterion corresponds to an optimum path for the min-max criterion in the graph with weights

$$y_e^{(r)} = \frac{w_e^{(r)}}{sp_r},$$

for all $e \in E$ and $r \in S$. Approximation algorithms for the former criterion can be obtained from algorithms for the latter by considering the modified weights instead of the original ones.

**Interval data.** Another natural model to describe uncertainty uses closed intervals in place of edge weights. Every edge $e \in E$ is equipped with two values $l_e$ and $u_e$ with $0 < l_e \leq u_e$ and we are guaranteed that $w_e \in [l_e, u_e]$ in any realization. It is easy to see that in this setting a robust path according to the min-max criterion simply corresponds to a shortest path in the realization where all weights are set to the
upper bound of the respective interval. Concerning the computation of robust paths according to the min-max relative criterion, Averbakh and Lebedev [10] and Zieliński [132] independently prove strong and weak NP-hardness, respectively. Kasperski and Zieliński [76] further study the approximation of this problem. They show that for this setting the approximation ratio of the mid-point solution is 2 and how to construct an FPTAS for those cases for which a pseudo-polynomial time algorithm exists (for example, for series-parallel graphs [77]).

The problem of computing an optimum path according to the min-max relative regret criterion has not received as much attention as the non-relative one, most likely because the two problems appear very closely related and therefore not worthy of an in-depth separate study. Averbakh [9] however shows that the relative case is indeed as hard as the non-relative one, and that many results for the latter apply to the former as well.

Montemanni and Gambardella [103] propose a generalization allowing the introduction of dependencies between edge weights. Given a matrix $M \in \mathbb{R}^{m \times m}_+$ and a vector $b \in \mathbb{R}^m_+$, each realization is a point in the corresponding convex polytope, i.e., a vector with weights $w = \left( w_{e \in E} \right) \in \mathcal{F} = \{ y \mid My \leq b \}$. An optimum path according to the min-max criterion can be written as

$$\arg \min_x \max_{w \in \mathcal{F}} \left\{ \sum_{e \in E} x_e \cdot w_e \right\}$$

subject to (1.1) and (1.2).

If $sp_w$ denotes the weight of a shortest path in the realization where the weights are $w$, we can express an optimum path according to the min-max regret criterion as

$$\arg \min_x \max_{w \in \mathcal{F}} \left\{ \sum_{e \in E} x_e \cdot w_e - sp_w \right\}$$

subject to (1.1) and (1.2).

The original interval data setting can be obtained by imposing the matrix $M$ to define box constraints. Montemanni and Gambardella [103] show the problem of computing a robust path according to the min-max criterion to be NP-hard, while NP-hardness for the min-max regret criterion follows straightforwardly from the interval data setting.
**Γ-intervals.** Bertsimas and Sim [17] develop a framework for optimization under uncertainty based on the observation that, in practical applications, it is unlikely for all the edge weights to be uncertain at the same time. Applied to the shortest path problem, the framework assumes that the weight of each edge $e \in E$ is uncertain and can deviate from a nominal weight $w_e$ to any value within $[w_e, w_e + d_e]$, for $d_e \geq 0$. The objective is a path minimizing its maximum weight when at most a fixed number $\Gamma$ of edges are allowed to deviate from the nominal weight. We can write this path as

$$\arg \min_x \sum_{e \in E} x_e \cdot w_e + \max_{S \in \Phi} \sum_{e \in S} x_e \cdot d_e$$

subject to (1.1) and (1.2).

where $\Phi = \{S \subseteq E \mid |S| \leq \Gamma\}$.

Bertsimas and Sim [16] prove that an optimum path for the above program can be computed in polynomial time by solving at most $m + 1$ additional programs. For each $d_l \in \{d_e \mid e \in E\} \cup \{0\}$, we let $x_i^*$ be

$$\arg \min_x \sum_{e \in E} x_e \cdot (w_e + \max\{d_e - d_l, 0\}) + \Gamma d_l$$

subject to (1.1) and (1.2).

An optimum path is then selected as one with minimum weight among all $x_i^*$. Since the objective functions of the additional problems are linear, each $x_i^*$ is the solution of a usual shortest path problem instance, and thus it can be computed in polynomial time.

We observe that this model is a generalization of the interval data model presented above; the latter can be obtained from the former by setting $\Gamma = m$. The efficient solution presented above computes, according to our categorization, an optimum path for the min-max criterion. Computing an optimum path for the min-max regret in this setting is at least as hard as for the interval data model, and thus strongly NP-hard.

**bw-robustness.** Roy [120] proposes a model allowing the user to control the degree of conservatism of the robust solution. In this setting, the user specifies two values $b$ and $w$, with $0 \leq b \leq w$. The goal is to compute a solution whose weight is never more than $w$ in any realization while the number of realizations where its weight is smaller or equal
than \( b \) is maximized. Gabrel et al. [56] apply this model to the shortest path problem and show that computing a robust path in this setting is \( \text{NP} \)-hard in the cases where the realizations are given by a discrete set of scenarios or in the form of an interval set as in the interval data setting.

### 1.3 Recoverable Robustness

Liebchen et al. [88] merge concepts from non-adaptive and adaptive models in a two-stage model. The first stage is a planning phase, where we compute a solution that might become sub-optimum when a realization is revealed. The second stage is a recovery phase, where the actual realization is revealed and we are allowed to modify parts of the current solution in order to improve it or make it feasible again. However, in the recovery phase we cannot modify the chosen solution arbitrarily, but we have to follow some constraints. Büsing [29] proposes two different applications of the concept of recoverable robustness to the shortest path problem.

In the first application we are given a first-stage weight \( w_e^{(1)} \) for every edge \( e \in E \) and a set \( \mathcal{F} \subset \mathbb{R}_+^m \) representing all the possible realizations (i.e., the second-stage weights). Once a path for the first stage is chosen, a realization is revealed and we can select another path. The constraint is that the first- and the second-stage path differ by at most \( k \) edges. Denoting the difference in number of edges between two paths \( x, x' \) as \( |x - x'| \), we can formulate this problem as

\[
\arg \min_{x, x'} \sum_{e \in E} x_e \cdot w_e^{(1)} + \max_{w^{(2)} \in \mathcal{F}} \sum_{e \in E} x'_e \cdot w_e^{(2)}
\]

subject to \((1.1)\) and \((1.2)\), 
\[|x - x'| \leq k.\]

Büsing [29] considers the cases where \( \mathcal{F} \) is: a discrete set of scenarios, an interval set, or a \( \Gamma \)-interval set. In all these cases, the problem turns out to be \( \text{NP} \)-hard.

In the second application, the first-stage weights represent rent costs. In the second stage we are allowed to choose any other path, but if we reuse an edge that was rented in the first stage we get a discount for it. We however have to pay a penalty for each new edge we acquire. More
formally, we are given a set of realizations $F \subset \mathbb{R}_+^m$ and two parameters $\alpha \in [0, 1]$ and $\beta \geq 0$. The first stage weight of $e$ in a realization $w \in F$ is $\alpha \cdot w_e$. In the second stage, the weight of an edge that was selected in the first stage is $(1 - \alpha) \cdot w_e$, but the weight of an edge that was not rented in advance is $(1 + \beta) \cdot w_e$. We look for two paths $x$ and $x'$ with smallest first- and second-stage weight in all possible realizations. We can express these paths as

$$\arg \min_{x, x'} \max_{w \in F} \left\{ \alpha \sum_{e \in E} x_e \cdot w_e + (1 - \alpha) \sum_{e \in E} x'_e \cdot w_e + (\alpha + \beta) \sum_{e \in E} x'_e (1 - x_e) \cdot w_e \right\}$$

subject to (1.1) and (1.2).

The cases where $|F| = 1$ or where $\alpha \in \{0, 1\}$ can be solved trivially because the first- and second-stage paths are the same. Büsing [29] shows the problem to be NP-hard in the case where $F$ is a discrete set of scenarios, or a $\Gamma$-interval set. In the special case where $\Gamma = m$, the problem can be solved in polynomial time by considering the single scenario where every edge weight is set to the upper bound of the respective interval.

1.4 Stochastic Models

The previous models assume that the only information available on the uncertain data is the range of values it can attain. For some applications we might however expect to have more refined knowledge on the uncertainty. In particular, we may know how likely it is for each realization to actually appear in reality. It might therefore not be interesting to optimize according to the worst-case scenario, if it appears with low probability. In these cases we are usually more interested in computing a path minimizing its expected weight, or some combination of its expected weight and its variance. In the following, we briefly review the most prominent adaptive and non-adaptive stochastic models that have been applied to the shortest path problem.

Stochastic Canadian Traveler Papadimitriou and Yannakakis [114] propose a variant of the CTP where the weight of each edge is given
in form of a discrete probability distribution, and its actual weight is discovered only when one of its endpoints is reached. They prove that the problems of computing an $s$-$t$ path with minimum expected competitive ratio or minimum expected weight are \#P-hard. Fried et al. [54] strengthen the latter result by showing it to be PSPACE-complete. Nikolova and Karger [110] provide efficient exact algorithms for the problem in the special cases where the graph is directed acyclic or all the $s$-$t$ paths are vertex-disjoint.

**Stochastic Shortest Path** If we assume the weight of an edge $e \in E$ to be a random variable $W_e$, the weight of a path $x$ is a random variable $W_x = \sum_{e \in E} x_e \cdot W_e$ as well. The main criteria of optimization for non-adaptive stochastic models are the expected weight of a path and its variance (or its standard deviation). The first criterion represents the expected travel time of a path; clearly, we wish to keep this value reasonably small. The second criterion indicates how much the travel time of a path can deviate from its expected value. Borrowing terminology from the language of economics, we denote users that prefer to keep the variance small at the cost of a more elevated expected weight as risk averse, while users preferring the opposite are denoted as risk seeking.

In its extreme case, a risk seeking user does not care about the variance of a path and wishes to minimize only its expected weight. If the mean value of each edge weight $\mu_e$ is known or it can be computed, the expected weight of a path $x$ is a linear function $E[W_x] = \sum_{e \in E} x_e \cdot \mu_e$. An optimum path can then be computed using classical shortest path algorithms, regardless of whether the edge weights are independent or not. The other extreme case, that of a risk averse user wishing to minimize only the variance of a path and not its expected weight, can be computed as easily only if the edge weights are uncorrelated (and thus the sum of the variances is a linear function).

Inspired by a common approach in economics to deal with risk aversion, Nikolova [108] proposes the computation of a path minimizing a linear combination of mean and standard deviation. Assuming to be given an estimation $\alpha \geq 0$ of the user’s risk aversion and for each edge $e \in E$ the
mean $\mu_e$ and variance $\tau_e$ of its weight $W_e$, we look for a path

$$\arg\min_x \sum_{e \in E} x_e \cdot \mu_e + \alpha \sqrt{\sum_{e \in E} x_e \cdot \tau_e}$$

subject to (1.1) and (1.2).

The problem of optimizing a concave objective function as in the above formulation is in general hard to approximate even within a logarithmic factor [79]. Nikolova et al. [111] propose an exact algorithm that finds a solution by enumerating all paths minimizing a linear combination of mean and variance (and not standard deviation) and show that, for the case of the shortest path problem, the number of paths considered is surprisingly only at most $n^{O(\log n)}$, that is, sub-exponential. Nikolova [109] further proves the problem to admit an FPTAS.
Chapter 2

Routing via Approximation Sets

In this chapter we consider a general framework for robust optimization in the presence of uncertainty [28] and apply it to the task of route planning for private vehicles. Let for this purpose $G = (V, E)$ be a directed graph with time-dependent edge weights $w : E \times T \rightarrow \mathbb{N}$ defined over a given time horizon $T$. The graph represents a road network in which $w(e, \tau)$ expresses the time needed to traverse the road $e \in E$ when a vehicle enters $e$ at time $\tau \in T$. A path is a sequence $P = \langle v_1, \ldots, v_k \rangle$ of vertices with $(v_i, v_{i+1}) \in E$ for $i = 1, \ldots, k - 1$. We overload the function $w$ to express the travel time of $P$ departing at $\tau \in T$ as

$$w(P, \tau) = \begin{cases} 0 & \text{if } k = 1, \\ w((v_1, v_2), \tau) + w(P', \tau + w((v_1, v_2), \tau)) & \text{otherwise,} \end{cases}$$

where $P'$ is the path obtained from $P$ without the last hop. Given a source $s \in V$, a target $t \in V$, and a time $\tau \in T$, the problem we consider is the robust counterpart of the quickest path problem, that asks for an $s$-$t$-path $P$ minimizing $w(P, \tau)$. Note that in the above definition we do not allow waiting at vertices; this is clearly not beneficial if waiting at a vertex on an $s$-$t$ path does not result in arriving at $t$ earlier, a property formally defined as follows.

**Definition 2.1.** A weight function $w : E \times T \rightarrow \mathbb{N}$ satisfies the FIFO
property if for all $e \in E$ and all $\tau, \tau' \in T$ with $\tau \leq \tau'$ it holds $\tau + w(e, \tau) \leq \tau' + w(e, \tau')$.

If the edge weights satisfy the FIFO property, a quickest path can be computed efficiently using a generalization of Dijkstra’s algorithm [52]. Since road networks are known to satisfy this property, we therefore focus on algorithms producing solutions not waiting at nodes.

### 2.1 Approximation Sets Framework

The approach by Buhmann et al. [28] assumes that an unknown problem generator $\Psi$ generates related instances that differ due to noise. Nothing is known about the noise or $\Psi$ itself and all we are given are two instances $I_1$ and $I_2$ generated by it. Our goal is to compute a path that is likely to be good for a future yet unknown instance $I_3$ from $\Psi$. For example, $I_1$ and $I_2$ represent the travel times according to the traffic situation of last Monday and Monday two weeks ago, while $I_3$ represents next Monday. For this purpose, we let an instance $I_i$ specify an edge weight function $w_i : E \times T \to \mathbb{N}$.

Since nothing is known about the underlying noise, a natural choice is to consider only paths that are good both for $I_1$ and $I_2$. From the set of all $s$-$t$ paths $\mathcal{P}$ we compute the approximation sets $A_\rho(I_1)$ and $A_\rho(I_2)$.
defined, for \( i \in \{1, 2\} \) and \( \rho \geq 1 \), as

\[
A_\rho(I_i) = \{ P \in \mathcal{P} \mid w_i(P, \tau) \leq \rho \cdot OPT_i \},
\]

\[
OPT_i = \min_{P \in \mathcal{P}} w_i(P, \tau).
\]

To avoid trivialities we assume that \( OPT_i > 0 \) for all \( i \in \{1, 2\} \). A robust path is picked uniformly at random from the intersection of the two approximation sets for a suitable choice of \( \rho \). Figure 2.1 shows an example of approximation sets and their intersection. To successfully apply the approach, the following issues need to be addressed:

(a) The definition of an appropriate “suitable” value \( \rho \).

(b) The computation of the intersection of the approximation sets.

(c) The random selection of a path from the intersection.

Ideally, we want a suitable \( \rho \) to be effective, i.e., producing paths of good quality, and computationally easy, i.e., allowing these paths to be computed efficiently. Needless to say, these two requirements are often contradicting. Also, it is usually preferable to avoid an explicit generation of the approximation sets and their intersection. In many cases, for example if these sets have exponential size, we would rather get the desired results without the effort of enumerating through a huge number of solutions.

In the following we study these issues in the context of routing for private vehicles. In Section 2.2 we consider item (a) and inspect two possible choices for \( \rho \), denoted as maximum similarity and first intersection. We compare the effectiveness of these choices via an experimental evaluation on a road network of the Berlin and Brandenburg area, kindly provided by the company TomTom within the project eCOMPASS [1]. In Section 2.3 we consider the speed-up item (b) for the first intersection method and engineer a bi-directional algorithm that is quite efficient on real-world instances. Aspects of item (c) are looked at in Chapter 3, where we apply the Markov chain Monte Carlo technique to the sampling of \( s-t \) paths in planar graphs.

### 2.2 Maximum Similarity & First Intersection

Looking for a suitable \( \rho \), Buhmann et al. [28] define the concept of similarity between two instances \( I_1, I_2 \) produced by \( \mathcal{P}\mathcal{G} \). Roughly speaking,
the similarity is a measure indicating, for any $\rho \geq 1$, the ratio of robust solutions over non-robust ones in $A_{\rho}(I_1) \cap A_{\rho}(I_2)$. According to this perspective, the authors therefore define as suitable any $\rho$ maximizing the similarity. A complication in this framework is that the similarity of two instances for a fixed $\rho$ is not a trivial quantity to calculate. In particular, its definition contains a term called the expected size of the intersection. Even though it has been shown that for some problems this term can be computed efficiently, for the shortest/quickest path problem the same does not seem likely; it is not even known whether for this problem case a compact formulation of the term exists (i.e., a formulation depending on a polynomial number of factors). For this reason, we consider one of the several properties shown by Buhmann et al. [28], that is the fact that the similarity is lower bounded by

$$\frac{|A_{\rho}(I_1) \cap A_{\rho}(I_2)|}{|A_{\rho}(I_1)| \cdot |A_{\rho}(I_2)|'}$$

and use this ratio as an estimate of the similarity. We thus seek a value of $\rho$ maximizing eq. (2.1). To avoid new terminology, in the following we denote the method computing a robust path from the intersection corresponding to the maximum value of eq. (2.1) as maximum similarity. The first intersection method computes instead a robust path from the intersection of the approximation sets for the smallest value of $\rho$ for which eq. (2.1) is nonzero.

Routing according to the maximum similarity or the first intersection methods is in general NP-hard. Computing the size of one approximation set for a fixed $\rho$ is $\#P$-hard already for the non-time dependent case [126] and, to the best of our knowledge, no exact, approximate or randomized algorithms with sub-exponential run-time are known. In order not to mix computational and evaluation aspects, we separate the tasks of assessing the quality of these methods from that of providing efficient algorithms. For the remainder of this section we consider the former task and provide an experimental evaluation of the methods on a real-world road network using an algorithm with worst-case exponential running time.

**Enumeration of paths.** We briefly review an algorithm by Rizzi et al. [118] listing all simple s-t paths of weight at most $L$ in a graph with static edge weights in a recursive, DFS-like manner. The argument at each recursive call is an s-u path $P$ for some $u \in V$ (starting with $P = \langle s \rangle$). If $u = t$, we report a new s-t path and return. Otherwise, we
consider the neighbors of \( u \) not already on \( P \). For each such neighbor \( v \), we compute the weight \( \alpha \) of a shortest \( v \)-\( t \) path. If \( w(P) + w(u, v) + \alpha \leq L \), we recur on the path obtained by concatenating \( v \) at the end of \( P \).

We generalize the algorithm from Rizzi et al. by replacing the static edge weights with time-dependent ones. We also assume an upper bound \( \rho_{\text{max}} \) on the value of \( \rho \) maximizing eq. (2.1) is known. This bound can be obtained by empirically investigating the available data; for our case, it turned out that a suitable value for \( \rho_{\text{max}} \) is 1.1. At the beginning, we compute the weights \( \sigma_1, \sigma_2 \) of a quickest path in each of the two instances. During the computation, we recur on \( v \) if the sum of the weights of the current \( s \)-\( v \) path and of the quickest path from \( v \) to \( t \) is at most \( \rho_{\text{max}} \cdot \sigma_i \) for at least one \( i \in \{1, 2\} \).

We can use the generalized algorithm to compute \( |A_{\rho}(I_1)|, |A_{\rho}(I_2)|, |A_{\rho}(I_1) \cap A_{\rho}(I_2)| \) and evaluate eq. (2.1) for all \( \rho \in [1, \rho_{\text{max}}] \). We cannot however store the sizes of the approximation sets and their intersection explicitly, since this might require non-polynomial space. To overcome this issue, we discretize the interval \([1, \rho_{\text{max}}]\) and approximate the maximum value of eq. (2.1) according to this discretization. For each \( A_{\rho}(I_i), i \in \{1, 2\} \), we introduce a sequence of counters \( c_0, \ldots, c_{K-1} \). Each \( c_j \) stores the number of paths \( P \) such that \( j \gamma \cdot \text{OPT}_i < w_i(P, \tau) \leq (j + 1) \gamma \cdot \text{OPT}_i \), with \( \gamma := \rho_{\text{max}} / K \). Every time a new \( s \)-\( t \) path \( P \) is found, we evaluate its weight in the two instances and increment the corresponding counters for the two approximation sets. We introduce counters and update them accordingly also for \( |A_{\rho}(I_1) \cap A_{\rho}(I_2)| \). The size of an approximation set or the intersection for a fixed \( \rho \) is computed by summing up all counters \( c_j \) with \( j \gamma \leq \rho \). At the end of the computation we select the value \( \rho^* \in \{j \gamma \mid j = 0, \ldots, K - 1\} \) maximizing eq. (2.1). Note that \( \rho^* \) is however only an approximation of the optimum \( \rho \); the accuracy of this approximation is directly proportional to the number \( K \) of counters used.

We now consider the task of selecting a path uniformly at random from \( A_{\rho^*}(I_1) \cap A_{\rho^*}(I_2) \). Instead of explicitly storing all the exponentially many \( s \)-\( t \) paths contained in the intersection for all \( \rho \in [1, \rho_{\text{max}}] \), we keep only \( K \) paths \( P_0, \ldots, P_{K-1} \), for the same value \( K \) as above. Each \( P_i \) has the property that at the end of the computation it holds that \( P_i \) is selected uniformly at random from \( A_{i \gamma}(I_1) \cap A_{i \gamma}(I_2) \). To compute these paths, whenever an \( s \)-\( t \) path \( P \) is found, we also calculate the smallest \( \rho \in [1, \rho_{\text{max}}] \) for which \( P \) belongs to the intersection of the two approximation sets, and the smallest \( i \) such that \( \rho \leq i \gamma \) (if there exists
Listing 2.1: The recursive enumeration algorithm.

Enumerate \((P = \langle s, \ldots, u \rangle)\)

\[\text{if } u = t \text{ then } \{s-t \text{ path found}\}\]

\(l := 0\)

\[\text{for } i \in \{1, 2\} \text{ do}\]

\[j := \left\lceil \frac{w_i(P, \tau)}{\text{OPT}_i} \cdot \frac{1}{\gamma} \right\rceil - 1\]

\[\text{if } j < K \text{ then } c_j^{(i)} := c_j^{(i)} + 1\]

\[\text{if } l < j \text{ then } \text{max} := j\]

\[\text{if } l < K \text{ then}\]

\[c_l^n := c_l^n + 1, \quad \delta := \sum_{j=0}^{l} c_j^n\]

\[P_l := P \text{ with probability } \frac{1}{\delta}\]

\[\text{else } \{\text{recursive step}\}\]

\[\text{for } e = (u, v) \in E \text{ do}\]

\[P' := \langle s, \ldots, u, v \rangle, \quad \text{rec} := \text{false}\]

\[\text{for } i \in \{1, 2\} \text{ do}\]

\[\tau' := \tau + w_i(P', \tau)\]

\[\alpha := \langle \text{weight quickest } v-t \text{ path departing at } \tau'\rangle\]

\[\text{if } w_i(P', \tau) + \alpha \leq \rho_{\text{max}} \cdot \sigma_i \text{ then } \text{rec} := \text{true}\]

\[\text{if } \text{rec} = \text{true} \text{ then Enumerate}(P')\]

no such \(\rho\) we ignore the path). After updating the counters, we iterate through all \(j \in \{i, \ldots, K-1\}\). For each such \(j\) we compute \(\delta = \sum_{k=0}^{j} c_k\) and set \(P_j = P\) with probability \(1/\delta\).

The pseudocode of the enumeration algorithm is shown in Listing 2.1. We use \(c_j^{(i)}\) and \(c_j^n\) to denote the \(j\)-th counter respectively of the approximation set \(A_\rho(I_i)\) and of the intersection, for \(i \in \{1, 2\}\).

**Lemma 2.2.** At the termination of the enumeration algorithm, for all \(i \in \{0, \ldots, K-1\}\) and \(P \in A_{i\gamma}(I_1) \cap A_{i\gamma}(I_2)\) it holds

\[\mathbb{P}[P_i = P] = \frac{1}{|A_{i\gamma}(I_1) \cap A_{i\gamma}(I_2)|} .\]
Proof. Let $\delta = \sum_{j=0}^{i} c_j$ at the time when $P$ is discovered. Since $P_i$ is set to $P$ with probability $1/\delta$ and at termination $P_i = P$ only if $P$ was never replaced by another path, it holds

$$
P[P_i = P] = \frac{1}{\delta} \cdot \frac{\delta}{\delta + 1} \cdots \frac{|A_{i\gamma}(I_1) \cap A_{i\gamma}(I_2)| - 1}{|A_{i\gamma}(I_1) \cap A_{i\gamma}(I_2)|} \cdot \frac{1}{|A_{i\gamma}(I_1) \cap A_{i\gamma}(I_2)|}.
$$

\hfill \square

**Computational Results**

We performed an experimental evaluation of the maximum similarity and the first intersection methods on a road network of the Berlin and Brandenburg area, kindly provided by the company TomTom within the EC project eCOMPASS [1]. The experimental evaluation was performed on one core of an Intel Xeon E5-2697v2 processor clocked at 2.7 GHz and 64 GB main memory. The code was written in C++ and compiled using GNU C++ compiler version 4.8.2 and optimization level 3.

**Input.** The travel time $w(e, \tau)$ needed to traverse the edge $e \in E$ at time $\tau \in T$ is obtained from a piecewise-linear function $f_e : T \rightarrow \mathbb{N}$ with period of one week, represented succinctly by a number of breakpoints. Each function $f_e$ is constructed as an aggregation of repeated measurement of the travel time on the street corresponding to $e$ for a period spanning several months\(^1\). Additionally, each edge $e \in E$ is provided with a set of measured travel times $T_e$. Each element $(\delta, \omega) \in T_e$ is called a *probe* and indicates the travel time $\omega$ on $e$ for an *absolute* point in time $\delta$. A probe is a more accurate measurement of the travel time than the function $f_e$ and it is therefore preferred to it. However, probes are very expensive to collect and store and thus are available only in a very limited amount. In our computation, we assume each probe to be valid for a period $\epsilon$ of 5 minutes, and to replace the underlying function $f_e$ in its period of validity. To summarize, the weight of $e \in E$ for a given $\tau \in T$ is calculated as

$$
w(e, \tau) = \begin{cases} 
\omega & \text{if } \exists (\delta, \omega) \in T_e : \delta \leq \tau < \delta + \epsilon, \\
 f_e(\tau) & \text{otherwise.}
\end{cases}
$$

\(^1\)We do not have complete knowledge on the way in which these functions are computed, because the algorithm is property of the company TomTom and not public.
In the TomTom dataset, a week is divided into 2016 breakpoints, one every 5 minutes. We can therefore compute $w(e, \tau)$ for any $\tau \in T$ in time $O(1)$.

The largest strongly connected component of the road network consists of 443,365 vertices and 1,038,284 edges. The underlying functions $f_e$ are constant for 750,544 edges and piecewise-linear as above for the remaining ones. Each edge of the graph is equipped with a set of probes distributed in a period of almost two weeks, from March 18th 2012 to March 31st 2012. The average number of probes per edge within these two weeks is 32.

**Setup.** To assess the quality of the maximum similarity and the first intersection methods, we consider two instances $I_1$ and $I_2$ corresponding to travel times in two different days. We define the quality of a robust route as its travel time in a future day $I_3$. Ideally, we would like to select $I_1$ and $I_2$ that are as much as possible correlated to $I_3$. For example, $I_1$ and $I_2$ are the travel times for respectively last Monday and Monday two weeks ago, while $I_3$ are the travel times for next Monday. However, our dataset does not allow us this particular choice, because the probes information we have available covers a period of less than two weeks. Another viable choice is to select $I_1$ and $I_2$ as the travel times respectively for today and for yesterday and evaluate the computed routes for the travel times $I_3$ of tomorrow. We can implement such a choice by selecting three consecutive days within the period between March 18th 2012 and March 31st 2012. Given that we require as much correlation as possible between $I_1$, $I_2$ and $I_3$, we select consecutive days in the middle of the week. We thus set $I_1$ to Tuesday March 20th, $I_2$ to Wednesday 21st, and $I_3$ to Thursday 22nd. To complete the setup, we pick departure times $\tau_1$, $\tau_2$ and $\tau_3$ respectively in $I_1$, $I_2$ and $I_3$. Since robust routes are most needed in periods of high traffic congestion, we set these departure times at 17:00 in each of these days. In other words, paths in $I_1$ departs at $\tau_1$, in $I_2$ at $\tau_2$, and in $I_3$ at $\tau_3$.

The start and target vertices could be picked uniformly at random among all vertices. However, the Berlin and Brandenburg road network is too big to allow explicit enumeration of all the $s$-$t$ paths for any pair $s$ and $t$. For this reason, we consider only a sub-graph of the original network and pick $s$ and $t$ uniformly at random within this sub-graph. An additional benefit of this choice is that we can assess the quality of the compared methods according to different portions of the road network. We consider a sub-graph corresponding to a strongly
Table 2.1: Average quality of a robust path in $I_3$.

<table>
<thead>
<tr>
<th>Method</th>
<th>Without probes Average</th>
<th>Std.dev.</th>
<th>With probes Average</th>
<th>Std.dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVG</td>
<td>city center</td>
<td>1.0853</td>
<td>0.1786</td>
<td>1.0801</td>
</tr>
<tr>
<td></td>
<td>suburbs</td>
<td>1.0486</td>
<td>0.0988</td>
<td>1.0493</td>
</tr>
<tr>
<td>AGG</td>
<td>city center</td>
<td>1.0886</td>
<td>0.1963</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>suburbs</td>
<td>1.0465</td>
<td>0.0985</td>
<td>-</td>
</tr>
<tr>
<td>SIM</td>
<td>city center</td>
<td>1.0403</td>
<td>0.0732</td>
<td>1.0407</td>
</tr>
<tr>
<td></td>
<td>suburbs</td>
<td>1.0353</td>
<td>0.0579</td>
<td>1.0372</td>
</tr>
<tr>
<td>INT</td>
<td>city center</td>
<td>1.0849</td>
<td>0.1785</td>
<td>1.0791</td>
</tr>
<tr>
<td></td>
<td>suburbs</td>
<td>1.0509</td>
<td>0.1085</td>
<td>1.0518</td>
</tr>
</tbody>
</table>

connected component located in the center of Berlin (with 4,836 vertices and 12,266 edges) and one located in the suburbs of the city (with 4,432 vertices and 11,612 edges). The average number of probes per edge respectively for the center and for the suburbs of Berlin is 163 and 59.

**Results.** We measure the quality of a robust path as the ratio of its weight over the weight of a quickest path in $I_3$. Displayed in Table 2.1 are the average and standard deviation of this ratio, among all the performed experiments. The methods labeled SIM and INT correspond respectively to the maximum similarity and the first intersection. The method AVG corresponds to the one where a robust path is computed as a quickest $s$-$t$ path in the graph with weights

$$w(e, \tau) = \frac{w_1(e, \tau) + w_2(e, \tau)}{2}.$$

Finally, AGG is the method where a robust path is computed as a quickest path in a standard Thursday, that is, where the weight of an edge $e$ is calculated using only the underlying function $f_e$.

We performed two kinds of tests. In the first kind, the probes associated with an edge are not considered while computing a robust route; in the second kind, probes are considered (the quality of a path in $I_3$ is however always evaluated using probes). Our goal is to show that the use of more accurate measurements on the travel time is beneficial for the computation of robust routes, even if these accurate measurements are available only in a limited amount.
Table 2.1 seems to indicate that our claim is true for the sub-graph located in the center of Berlin and the methods AVG and SIM. These methods result in lower average and standard deviation in quality when probes are used in the calculation of the travel times. For the method SIM the improvement is only in the standard deviation and not in the average. However, the worsening in average when considering probes is very small, while the improvement in the standard deviation is larger than the other methods. Since a robust route is typically considered a risk-averse one, that is one where the expected deviation from the average travel time is small, we consider this a quite positive result.

For the graph located in the suburbs of Berlin the three methods AVG, SIM, and INT all behave similarly: the average quality of the routes worsen and the standard deviation improves after the introduction of probes. This fact is somewhat to be expected, because the suburbs of a city are typically subject to smaller amounts of congestion than the city center. Furthermore, this smaller amount is usually very much localized in a small area, for example the exit of an highway (while for the city center traffic congestion is usually more equally spread in the whole network). Therefore, the benefit of robustness are very much evident for this localized congested area, while for the rest of the suburban network the information provided by the aggregate functions $f_e$ is already sufficiently accurate.

Remark. The values shown for the methods AVG, AGG, and INT are averaged among 1,000 tests for each entry of Table 2.1. The same thorough evaluation could not however be made for the method SIM because the enumeration algorithm is not fast enough. While we could amend this issue for INT by replacing the exponential time algorithm with the fast bi-directional search that we will present in the following section, we do not have such an alternative for SIM. We were therefore forced to run only a very limited number of tests for SIM, roughly 20 per entry of Table 2.1. This more limited experimental assessment is probably the reason behind the fact that the numbers in the table for the method SIM are much smaller than for AVG and INT. Among the performed experiments, only those for which the distance between the selected $s$ and $t$ is sufficiently small could terminate in a reasonable amount of time.
2.3 Speed-up of First Intersection

We now consider the fast computation of robust routes according to a generalization of the first intersection method called min-max relative regret. This method looks for a route minimizing the maximum normalized travel time in the worst-case instance. In our case, there are $N$ instances ($N$ past days), and the normalization is done with respect to the travel time of a quickest route in the respective day.

More formally, we assume to be given a departure time $\tau \in T$, vertices $s,t \in V$, and a set $\{I_1, \ldots, I_N\}$ of $N$ instances (or scenarios), each specifying a time-dependent edge weight function $w_i : E \times T \rightarrow \mathbb{N}$. The relative regret of an $s$-$t$ path $P$ in $I_i$ is the ratio between its weight and the weight $OPT_i$ of a quickest path in $I_i$. Our goal is to compute a path minimizing its maximum relative regret. In other words, we look for a path

$$\arg \min_P \max_i \left\{ \frac{w_i(P, \tau)}{OPT_i} \right\}.$$  \hspace{1cm} (2.2)

To observe that the min-max relative regret is a generalization of the first intersection method, it is sufficient to set $N = 2$ and note that the maximum relative regret of an optimum path for the former corresponds to the smallest value of $\rho$ for which eq. (2.1) is nonzero.

We already introduced the min-max relative regret criterion and the similar min-max and min-max regret criteria for a discrete set of scenarios in Chapter 1. We pointed out that these problems are NP-hard even for $N = 2$. Since a worst-case efficient algorithm is unlikely to be found, in the following we therefore consider efficiency only from a practical perspective. Even though our theoretical results can be generalized for any number $N$ of scenarios, for practical reasons we focus on the particular case $N = 2$. As additional motivation for considering a small number of scenarios, we observe that approaches based on the min-max or min-max (relative) regret criteria are of more interest in these cases. If the number of scenarios is large, approaches based on statistical analysis are typically more efficient and produce results of similar if not better quality.

In a straightforward implementation, we can compute an optimum path for Problem (2.2) by enumerating all $s$-$t$ paths in order of increasing travel times, alternatively for each instance, until the desired $s$-$t$ path is found. This implementation is however computationally extremely demanding and impractical. Thus, we observe that its output lies on
the Pareto front of the graph with appropriate \textit{time-dependent} and \textit{multi-criteria} edge weights induced by the \(N\) instances. We adapt a known algorithm of Hansen/Martins [66, 99] for computing such a front and apply the speed-up technique of \textit{bi-directional} search to it. We also prove that the bi-directional algorithm can be parametrized by a factor \(K\) in order to compute a \(K\)-approximate solution. We experimentally show that the speed-up of the bi-directional search over uni-directional is considerable. A preliminary evaluation for \(N = 3\) seems to indicate that this speed-up scales with the number of instances. Finally, we propose a simple modification to the bi-directional algorithm exploiting the correlation between the travel times of the different instances. To the best of our knowledge, our work is the first one to consider speeding-up the computation of shortest paths (in particular applying the technique of bi-directional search) in a setting where every component of the multi-criteria edge weights is time-dependent.

\section*{Relation to Bi-criteria Quickest Paths}

For the case of 2 scenarios, the instances \(I_1\) and \(I_2\) with edge-weight functions \(w_1\) and \(w_2\) induce a bi-criteria weight function \(w : E \times T \rightarrow \mathbb{N}^2\)

\[
w(e, \tau) = \begin{pmatrix} w_1(e, \tau) \\ w_2(e, \tau) \end{pmatrix}.
\] (2.3)
We overload the definition of \( w \) to express the weight of a path \( P \) as
\[
  w(P, \tau) = w(P', \tau) + \left( \frac{w_1(e, \tau + w_1(P', \tau))}{w_2(e, \tau + w_2(P', \tau))} \right),
\]
where \( P' \) is the path obtained from \( P \) without its last hop \( e \). Given two \( s-t \) paths \( P \) and \( P' \), we say that \( P \) dominates \( P' \) if \( w_i(P, \tau) \leq w_i(P', \tau) \) for all \( i \in \{1, 2\} \), with the inequality being strict for some \( i \). If two paths have the same weight in both components, they are said to be equivalent.

The Pareto front of a set of paths \( \mathcal{P} \) is the subset of all paths in \( \mathcal{P} \) that are not dominated by another path in \( \mathcal{P} \) (see Figure 2.2 for an example). For the sake of readability, in the following we will assume that no two paths are equivalent. It is well known \cite{6} that an optimum path for Problem (2.2) lies in the Pareto front \( \mathcal{F} \) of all \( s-t \) paths departing at \( \tau \in T \).

The following theorem proves a slightly stronger statement.

**Theorem 2.3.** Let \( \mathcal{F}_\rho \) be the Pareto front of all optimum paths of Problem (2.2). Then, \( \mathcal{F}_\rho \subseteq \mathcal{F} \).

**Proof.** Assume towards contradiction that there exists a path \( P \in \mathcal{F}_\rho \setminus \mathcal{F} \). Then, there is a path \( P' \notin \mathcal{F}_\rho \) dominating \( P \). For \( i \in \{1, 2\} \), we let
\[
  \rho'_i = \frac{w_i(P', \tau)}{\text{OPT}_i} \leq \frac{w_i(P, \tau)}{\text{OPT}_i} = \rho_i.
\]
Note that the optimum relative regret is \( \rho^* = \max\{\rho_1, \rho_2\} \) and that it holds \( \max\{\rho'_1, \rho'_2\} > \rho^* \). If \( \max\{\rho'_1, \rho'_2\} = \rho'_i \) for some \( i \in \{1, 2\} \) we get a contradiction, because
\[
  \rho'_i \leq \frac{w_i(P, \tau)}{\text{OPT}_i} \leq \rho^* < \rho'_i.
\]

Theorem 2.3 implies that an optimum path for Problem (2.2) can be computed by enumerating all paths in \( \mathcal{F} \) and picking one with smallest relative regret. Note that there may exist paths in \( \mathcal{F} \) that are not optima, typically the quickest paths in either of the two instances. It is straightforward to prove Theorem 2.3 also for the min-max absolute and the min-max deviation criteria, implying that the bi-directional search algorithm proposed in the second half of this section can be applied for those criteria as well. We further observe that the paths in \( \mathcal{F}_\rho \) might not be extreme points of the convex hull of \( \mathcal{F} \). This observation rules out
the possibility of adopting known algorithms for the computation of such points [32, 55, 57].

Remark. The definition in eq. (2.4) might appear unusual to a reader familiar with bi-criteria quickest path problems. In the literature it is more typically assumed that one of the two criteria of the weight of a path is its travel time while the other one is a cost depending on the travel time (for example, fuel consumption). Such a weight function can be written as

$$w(P, \tau) = w(P', \tau) + \left( \frac{w_1(e, \tau + w_1(P', \tau))}{w_2(e, \tau + w_1(P', \tau))} \right).$$  (2.5)

Note the difference in the time at which the second component is evaluated. Since our target application is robust routing, we need to consider different travel times for the same path and hence use the definition in eq. (2.4). Under similar assumptions on the FIFO property of the edge weights, our results can be generalized for eq. (2.5) as well.

We now turn to the computation of an optimum path for Problem (2.2) by means of time-dependent multi-criteria optimization. Our aim is to apply the speed-up technique bi-directional search to an algorithm by Martins for computing Pareto fronts and experimentally investigate the improvements to its running time on road networks. In spite of its relevance, the literature about the problem is scant, and not many practical algorithms are known. The most closely related work is by Batz and Sanders [13] that consider the computation of shortest paths in a graph with multi-criteria edge weights where only one of the components is time-dependent. A great amount of work has been however invested by the community into the speed-up of routing algorithms in settings where edge weights are either only time-dependent [12, 34, 82] or only multi-criteria [35, 50].

Martins’ algorithm. Hansen [66] introduces several variants of the bi-criteria shortest path problem and a pseudo-polynomial time algorithm to compute Pareto fronts in graphs with static non-negative bi-criteria edge weights. Martins [99] generalizes this algorithm to static edge weights with more than two criteria. His algorithm keeps a priority queue of temporary labels $Q$ and a set of permanent labels $\pi_u$ for every vertex $u \in V$. Each label $(u, \omega)$ represents a path from $s$ to $u$ with weight $\omega \in \mathbb{N}^k$ (for $k$ criteria); we write $P \in \pi_u$ to indicate that the label representing $P$ is in $\pi_u$. At the beginning every $\pi_u$ is empty, and a label
Listing 2.2: Algorithm TdMartins.

\[
\forall v \in V : \pi_v := \emptyset \\
Q.\text{insert}((s,0))
\]

\{
\text{Compute front}\n\}

\textbf{while } Q \neq \emptyset \textbf{ do}

\[
(u, \omega) := Q.\text{extract\_min}(\quad) \\
\text{for } e = (u, v) \in E \textbf{ do}
\]

\[
v := \omega + \left( \frac{w_1(e, t + \omega_1)}{w_2(e, t + \omega_2)} \right) \\
\text{if } \neg \pi_v.\text{dominates}(v) \text{ and } \neg \pi_t.\text{dominates}(v) \textbf{ then}
\]

\[
Q.\text{insert}(v, v) 
\]

(s,0) is created and put into Q. At each iteration the algorithm extracts from Q the smallest label (u, \omega) in lexicographical order and puts it into \pi_u. A new label (v,v) is then generated for each vertex v that can be reached from u, with v = \omega + w(u,v). If no label in \pi_v or \pi_t dominates the new one, it is inserted into Q and all labels corresponding to s-v paths that are dominated by (v,v) are removed from Q. The algorithm ends when Q is empty; at this point, \pi_t contains labels representing all paths in the Pareto front \mathcal{F}. By storing labels in Q and in all \pi_v in lexicographical order, we can implement the operations of extract minimum, insertion, and dominance checking to run, for the bi-criteria case, in logarithmic time. For a number of criteria larger than 2 it is currently not known how to efficiently implement the latter of these operations.

Gräbener et al. [60] provide an experimental evaluation of a straightforward time-dependent extension of Martins’ algorithm, shown in Listing 2.2 with the name TdMartins, on some publicly accessible networks. The correctness of this algorithm, well-known but included for completeness in the following theorem, depends crucially on the FIFO property; the analysis of the run-time follows trivially from the analysis by Hansen [66]. Hamacher et al. [64] consider the setting where the FIFO property does not hold, and provide algorithms computing the Pareto front for a given s-t pair as well as for the all-to-all variant.

\textbf{Theorem 2.4.} Let G = (V, E) be a graph with edge weights \( w : E \times T \rightarrow \mathbb{N}^2 \) as in eq. (2.3). If \( w_i : E \times T \rightarrow \mathbb{N} \) satisfies the FIFO property for every
Chapter 2. Routing via Approximation Sets

\(i \in \{1, 2\}, \text{TdMartins computes the Pareto front } \mathcal{F}.\)

Proof. The computed front \(\pi_t\) is not correct if there is a path in \(\mathcal{F}\) that is not in \(\pi_t\), there is a path in \(\pi_t\) that is not in \(\mathcal{F}\), or both. We consider only the first case, since the remaining two follow from the fact that if \(\pi_t\) contains at least the paths in \(\mathcal{F}\) then all other paths are dominated by those.

Suppose towards contradiction that there exists \(P \in \mathcal{F}\) such that \(P \notin \pi_t\). Consider the prefix \(P_{sv}\) of \(P\) from \(s\) to the first vertex \(v\) such that \(P_{sv} \notin \pi_v\), and the suffix \(P_{vt}\) of \(P\) from \(v\) to \(t\). We can express the weight of \(P\) as

\[
w(P, \tau) = w(P_{sv}, \tau) + \begin{pmatrix} w_1(P_{vt}, \tau + w_1(P_{sv}, \tau)) \\ w_2(P_{vt}, \tau + w_2(P_{sv}, \tau)) \end{pmatrix}.
\]

Since \(P_{sv} \notin \pi_v\), there exists another path \(P'_{sv}\) dominating it, and we can obtain an \(s\)-\(t\) path \(P'\) (not necessarily simple) by concatenating \(P'_{sv}\) and \(P_{vt}\). The weight of \(P'\) can be written as

\[
w(P', \tau) = w(P'_{sv}, \tau) + \begin{pmatrix} w_1(P_{vt}, \tau + w_1(P'_{sv}, \tau)) \\ w_2(P_{vt}, \tau + w_2(P'_{sv}, \tau)) \end{pmatrix}.
\]

Since \(P'_{sv}\) dominates \(P_{sv}\), we know that, for every \(i \in \{1, 2\}\), it holds that

\[w_i(P'_{sv}, \tau) \leq w_i(P_{sv}, \tau).\]

Since both \(w_1\) and \(w_2\) satisfy the FIFO property, we get that \(w(P', \tau)\) dominates \(w(P, \tau)\). This contradicts the assumption that \(P \in \mathcal{F}.\) \(\square\)

Corollary 2.5. The run-time of TdMartins is in \(O(nmW \cdot \log(nW))\), where

\[W = \min_{i \in \{1, 2\}} \left\{ \max_{e \in E} w_i(e, \tau) \right\} \cdot \log(nW).\]

Bi-directional search. Dijkstra’s algorithm for finding shortest \(s\)-\(t\) paths in the static single-criteria case gradually grows a shortest-path tree from \(s\). At any step, each vertex is in one of the following states: settled, discovered, or unreached. A vertex is settled if its distance from \(s\) is known exactly, it is discovered if only an upper bound on the distance is known, and otherwise it is unreached. At every iteration the algorithm introduces a new edge in the shortest path tree and sets its tail vertex as settled. The algorithm terminates when \(t\) is settled.
In the worst-case the tree contains all vertices, even though we are only interested in those on the shortest \( s-t \) path that is returned.

The idea behind the bi-directional search [58, 59] is to grow two trees rooted at \( s \) and \( t \) using Dijkstra’s algorithm alternatively from \( s \) and from \( t \). The execution from \( t \), called backward search, uses the edges of the reverse graph, i.e., the graph containing the edges of the original one in reverse direction. As soon as a vertex \( v \) is settled by both the forward and the backward search the algorithm terminates and a shortest \( s-t \) path is guaranteed to lie in the union of the so-far constructed shortest path trees (such a path might however not pass through \( v \)). Any alternation works correctly; a typical choice is to balance the number of iterations of the two searches.

For static multi-criteria edge weights one can apply the bi-directional search by replacing Dijkstra’s algorithm with Martins’. Since the goal is to compute the whole Pareto front of \( s-t \) paths (and not only a single path), the stopping criterion is however different. Demeyer et al. [37] show that terminating the computation when the sum of the point-wise minima of the forward and backward queues is dominated by the front computed so far ensures that the Pareto front is found. The point-wise minimum of a queue \( Q \), denoted as \( Q_{p\_min}() \), is the vector where each component is equal to the minimum among all labels in \( Q \) for the corresponding criterion.

When the edge weights are time-dependent, even in the single-criterion case, applying the bi-directional search is not straightforward anymore: the input consists of \( s, t \), and the departure time \( \tau \). Thus, we can grow a tree from \( s \) starting at time \( \tau \), but we do not know the time \( \tau' \) from which we shall start growing the tree from \( t \) – ideally, \( \tau' \) is the earliest arrival time at \( t \), but that is the number we wish to compute. A way to overcome this difficulty is to make the backward search static: for each edge \((v, u)\) of the reverse graph \( \hat{G} = (V, \hat{E}) \), use a static weight defined as

\[
\hat{w}((v, u)) = \min_{\tau \in T} \{w((u, v), \tau)\}.
\] (2.6)

Nannicini et al. [107] propose a bi-directional algorithm using the weights in eq. (2.6) working in three phases. In phase 1 the forward and backward search run alternatively until a vertex is discovered in both directions, resulting in an upper bound \( \mu \) on the weight of a quickest path. In phase 2 both searches continue until the distances of all the discovered vertices in the backward queue are at least \( \mu \). In phase 3 only the forward search continues, with the constraint that only
vertices that were settled by the backward search are considered. In the following we show how to apply this idea to the time-dependent multi-criteria case.

**Bi-directional Time-dependent Martins’ Algorithm**

A bi-directional algorithm for edge with weights that are both time-dependent and bi-criteria can be designed by straightforwardly combining the ideas of Demeyer et al. and of Nannicini et al. Doing this results in a three-phases search using Martins’ algorithm both from $s$ and from $t$, where the edge weights in the reverse graph are defined as in eq. (2.6) for both criteria. The termination condition of the backward search (i.e., the end of phase 2) is the stopping condition of Demeyer et al. As it turns out, however, this trivial algorithm can be improved considerably.

A critical observation to improve the straightforward algorithm is to note that in the backward direction our only interest is to identify vertices that might be on a Pareto optimal path. In other words, to determine whether or not the Pareto front of a given vertex contains at least one “promising” label. However, a label that is good for one criterion might not be good for the other one and we cannot know in advance which labels are promising. Our solution is to compute only the pointwise minima of the Pareto front $\pi_v$ of each $v \in V$.

If the only purpose of the backward search is to compute pointwise minima, then Martins’ algorithm is more than what is necessary. We can instead implement the backward search as two independent Dijkstra’s runs on the reverse graph for each criterion. We modify the three phases of the bi-directional algorithm according to this observation as follows.

For phase 2, suppose we have found (in some way during phase 1) a number of non-necessarily Pareto-optimal $s$-$t$ paths, and let $M$ denote the Pareto front of these paths, while $\overrightarrow{Q}$ is the forward queue and $\overleftarrow{Q}_1, \overleftarrow{Q}_2$ are the backward queues. Suppose further that at some point during the computation, the weight of a path in $M$ dominates

$$\beta := \overrightarrow{Q}.p_{\text{min}} + \left(\overleftarrow{Q}_1.\text{min} + \overleftarrow{Q}_2.\text{min}\right).$$
At this point, if a vertex $v$ has not been settled by both backward searches, then the weight of any $s$-$t$ path through $v$ is dominated by $\beta$ and therefore by a path in $M$ (we prove the correctness of this argument formally in the following). We can thus terminate phase 2 and the backward searches as soon as a path in $M$ dominates $\beta$.

For phase 3 consider the situation where the forward search created a label $(v, \omega)$ to insert into $\overrightarrow{Q}$. Let the vector of distances computed by the backward searches for $v$ be $v.d$, the value of the second term of $\beta$ (the minima of the backward queues) at the end of phase 2 be $\overleftarrow{\beta}$, and the minimum between $v.d$ and $\overleftarrow{\beta}$ in each component be $\theta$; that is, for $i \in \{1, 2\}$, we define $\theta_i := \min \{v.d_i, \overleftarrow{\beta}_i\}$. At the beginning of the computation $v.d_i$ is set to $\infty$ and at the end it holds that $v.d_i \leq \overleftarrow{\beta}_i$ if $v$ has been settled by the $i$-th backward search. If in phase 3 a path in $M$ dominates $\omega + \theta$ then no path with the same $s$-$v$ prefix as $\omega$ can be optimal. We can thus discard all labels $(v, \omega)$ for which $\omega + \theta$ is dominated by a path in $M$.

According to the above phases, the purpose of phase 1 is the computation of a suitable tentative front $M$. Intuitively, a tentative front is good if the domination of $\beta$ happens as early as possible, because less labels carry over to phase 3. We propose to terminate phase 1 as soon as a vertex $v$ with $\pi_v \neq \emptyset$ is discovered by both backward searches and set $M$ as the corresponding set of $s$-$t$ paths passing through $v$. This strategy has the advantage of being efficient while at the same time being simple to implement. We note however that it is easy to come up with different strategies; it is an interesting open question to identify an optimum one.

More in detail, the phases of the bi-directional algorithm are as follows:

**Phase 1** We let the forward and the backward search run alternatively. In the forward direction we use the time-dependent Martins’ algorithm. In the backward direction we use two independent runs of Dijkstra’s algorithm, one per criterion, using edge weights as in eq. (2.6). This phase ends as soon as a vertex $v$ with $\pi_v \neq \emptyset$ is discovered by both backward searches and set $M$ as the corresponding set of $s$-$t$ paths passing through $v$. At the termination of the phase we let $M$ be the Pareto front of the $s$-$v$ paths in $\pi_v$ concatenated with the $v$-$t$ paths discovered in the backward direction.

**Phase 2** Both the forward and the backward searches continue to run
Listing 2.3: Algorithm BiTdMartins.

\[ M := \emptyset, \ \phi := 1, \ \forall v \in V : \pi_v := \emptyset, v.d := (\infty) \]
\[ \overrightarrow{Q}.\text{insert}(s,0), \ \overrightarrow{Q_1}.\text{insert}(t,0), \ \overrightarrow{Q_2}.\text{insert}(t,0) \]

while $\overrightarrow{Q} \neq \emptyset$ do
  if $\phi = 3$ then $\leftrightarrow := \rightarrow$ else $\leftrightarrow \in \{\rightarrow, \leftarrow\}$
  \{terminate phases 1 or 2\}
  if $\phi = 1$ and $\exists v \in V : \pi_v \neq \emptyset$ and $v.d_1 \neq \infty$ and $v.d_2 \neq \infty$
  then $M.\text{insert}(<s-t \text{ paths through } v>), \ \phi := 2$
  if $\phi = 2$ and $M.\text{dominates}(\beta)$ then $\phi := 3$
  \{relax edges\}
  if $\leftrightarrow := \rightarrow$ then
  \( (u,\omega) := \overrightarrow{Q}.\text{extract}\_\text{min}() \)
  for $e = (u,v) \in \overrightarrow{E}$ do
    $v := \omega + (\omega_1 + (v.d + \omega_1) + \omega_2)_{\varepsilon_i}$
    if $\phi = 3$ and $M.\text{dominates}(v + \theta)$ then continue
    if $-\pi_v.\text{dominates}(v)$ and $-\pi_t.\text{dominates}(v)$ then
      $\overrightarrow{Q}.\text{insert}(v,v)$
  else for $i \in \{1,2\}$ do
    $u := \overrightarrow{Q_i}.\text{extract}\_\text{min}()$
    for $e = (u,v) \in \overrightarrow{E}$ do
      if $u.d_i + \hat{w_i}(e) < v.d_i$ then $\overrightarrow{Q_i}.\text{insert}(v,u.d_i + \hat{w_i}(e))$

as in phase 1, until a path in $M$ dominates $\beta$.

**Phase 3** Only the forward search continues, with the constraint that labels $(v,\omega)$ for which $\omega + \theta$ is dominated by a path in $M$ are ignored. This phase terminates when $\overrightarrow{Q}$ becomes empty.

The pseudocode of the algorithm under the name of BiTdMartins is illustrated by Listing 2.3. In the code, we use $\phi$ to denote the current phase and $\leftrightarrow$ to denote either the forward ($\leftrightarrow := \rightarrow$) or the backward search ($\leftrightarrow := \leftarrow$). The command $\leftrightarrow \in \{\rightarrow, \leftarrow\}$ selects the direction for the current iteration according to the alternation strategy; in our implementation we alternate between one iteration of the forward search and one
iteration for each backward search. Note that to check the termination condition of phase 1 it is not necessary to search through all vertices. It is sufficient to check whether the condition holds only for the vertex extracted at the current iteration.

In the algorithm of Nannicini et al. [107] phase 2 terminates when the upper bound \( \mu \) computed in phase 1 is at most the minimum of the backward queue. The authors proved that replacing this condition with one that, for a fixed parameter \( K \), checks whether \( \mu \) is at most \( K \) times the minimum of the backward queue results in an algorithm computing a \( K \)-approximate quickest path (i.e., a path with weight at most \( K \) times the weight of a quickest path). The following theorem shows that \textsc{BtDmartins} satisfies a similar property. A corollary of this theorem, obtained by setting \( K = 1 \), implies correctness of the algorithm in the exact variant.

**Definition 2.6.** Given \( \tau \in \mathcal{T} \) and \( K \geq 1 \), we say that a path \( P \) is a \( K \)-approximation of another path \( P' \) if \( w(P, \tau) \) dominates or is equivalent to \( K \cdot w(P', \tau) \). Given two sets of paths \( \mathcal{P} \) and \( \mathcal{P}' \), we say that \( \mathcal{P} \) is a \( K \)-approximation of \( \mathcal{P}' \) if every path in \( \mathcal{P}' \) is \( K \)-approximated by a path in \( \mathcal{P} \).

**Theorem 2.7.** For \( K \geq 1 \), if we terminate phase 2 when \( M \) dominates \( K \cdot \beta \) and in phase 3 we discard all labels \((v, \nu)\) for which \( K \cdot (\nu + \theta) \) is dominated by \( M \), then \textsc{BtDmartins} computes a \( K \)-approximation of \( \mathcal{F} \).

**Proof.** Assume there exists a path \( P \in \mathcal{F} \) not \( K \)-approximated by a path in \( \pi_t \). That is, for every \( P' \in \pi_t \) there is \( i \in \{1, 2\} \) such that

\[
K \cdot w_i(P, \tau) < w_i(P', \tau). \tag{2.7}
\]

Let \( P_{sv} \) be the prefix of \( P \) from \( s \) to the first vertex \( v \) such that \( P_{sv} \notin \pi_v \), and \( P_{vt} \) be the suffix of \( P \) from \( v \) to \( t \). Since \( P_{sv} \notin \pi_v \), there is a path in \( M \) dominating \( K \cdot (w(P_{sv}, \tau) + \theta) \). Let \( P' \) be either this path, if it belongs to \( \pi_t \), or a path in \( \pi_t \) dominating it otherwise. Note that for all \( i \in \{1, 2\} \) it holds \( \theta_i \leq \overline{w_i}(P_{vt}) \) since, if \( v \) was settled by the \( i \)-th backward search, then \( \theta_i = v.d_i = \overline{w_i}(P_{vt}) \) while, if \( v \) was not settled by the \( i \)-th backward search, then \( \theta_i = \overline{\beta_i} \leq \overline{w}(P_{vt}) \). Supposing without loss of generality that eq. (2.7) holds for \( i = 1 \) we obtain a contradiction, because

\[
K \cdot w_1(P, \tau) < w_1(P', \tau) \leq K \cdot (w_1(P_{sv}, \tau) + \theta_1) \\
\leq K \cdot (w_1(P_{sv}, \tau) + \overline{w_1}(P_{vt})) \leq K \cdot w_1(P, \tau).
\]
Corollary 2.8. BiTdMARTINS computes the Pareto front $\mathcal{F}$.

Note that the converse of Theorem 2.7 in general does not hold. There might be paths in $\pi_t$ not approximating a Pareto optimal path. Figure 2.3 shows an example of such a situation.

Computational Results

We performed an experimental evaluation of BiTdMARTINS on the Berlin and Brandenburg road network provided by TomTom. The same settings and machines as in the previous evaluation were used.

Setup. To obtain two instances (edge weight functions) $I_1$ and $I_2$ we consider departure times $\tau_1$ and $\tau_2$ in two consecutive days. We select uniformly at random one of the 24 hours of a day and let $\tau_1$ be the corresponding point in time on Tuesday and $\tau_2$ be the same time in the following Wednesday. The edge weight functions are obtained by setting the beginnings of the time horizon (in other words the departure times) of $I_1$ and $I_2$ respectively at $\tau_1$ and at $\tau_2$. We select 10,000 pairs of vertices $s$ and $t$ uniformly at random.

Results. Table 2.2 shows a comparison of the algorithms considered in terms of quality (i.e., the maximum relative regret of the computed
Table 2.2: Average relative regret, run-time, and number of scanned labels.

<table>
<thead>
<tr>
<th></th>
<th>Max Rel Regret</th>
<th>Run-time (ms)</th>
<th>Labels Phase 1</th>
<th>Labels Phase 2</th>
<th>Labels Phase 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dijkstra</td>
<td>1.0711</td>
<td>261</td>
<td>-</td>
<td>-</td>
<td>220,620</td>
</tr>
<tr>
<td>Naive</td>
<td>1.0074</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>8,420</td>
</tr>
<tr>
<td>Uni-dir</td>
<td>1.0074</td>
<td>3,105</td>
<td>-</td>
<td>-</td>
<td>1,419,524</td>
</tr>
<tr>
<td>Bi-dir</td>
<td>1.0074</td>
<td>1,888</td>
<td>178,855</td>
<td>189,336</td>
<td>449,560</td>
</tr>
<tr>
<td>K=1.02</td>
<td>1.0085</td>
<td>1,570</td>
<td>178,855</td>
<td>177,133</td>
<td>402,970</td>
</tr>
<tr>
<td>K=1.04</td>
<td>1.0108</td>
<td>1,427</td>
<td>178,855</td>
<td>165,209</td>
<td>356,479</td>
</tr>
<tr>
<td>K=1.06</td>
<td>1.0156</td>
<td>1,286</td>
<td>178,855</td>
<td>153,549</td>
<td>311,216</td>
</tr>
<tr>
<td>K=1.08</td>
<td>1.0232</td>
<td>1,150</td>
<td>178,855</td>
<td>142,190</td>
<td>268,139</td>
</tr>
<tr>
<td>K=1.10</td>
<td>1.0337</td>
<td>1,028</td>
<td>178,855</td>
<td>131,160</td>
<td>228,646</td>
</tr>
<tr>
<td>K=1.20</td>
<td>1.0724</td>
<td>712</td>
<td>178,855</td>
<td>80,678</td>
<td>93,660</td>
</tr>
<tr>
<td>K=1.40</td>
<td>1.0830</td>
<td>338</td>
<td>178,855</td>
<td>16,854</td>
<td>10,420</td>
</tr>
<tr>
<td>K=1.60</td>
<td>1.0856</td>
<td>275</td>
<td>178,855</td>
<td>1,858</td>
<td>2,407</td>
</tr>
<tr>
<td>K=1.80</td>
<td>1.0865</td>
<td>269</td>
<td>178,855</td>
<td>270</td>
<td>1,787</td>
</tr>
<tr>
<td>K=2.00</td>
<td>1.0868</td>
<td>269</td>
<td>178,855</td>
<td>81</td>
<td>1,692</td>
</tr>
</tbody>
</table>

path) and efficiency, averaged among the performed 10,000 tests. The efficiency of an algorithm is measured in terms of CPU time and the number of labels scanned for each phase of the algorithm. The number of labels scanned, i.e., the overall number of labels extracted from the forward and from the backward queues, represents a machine-independent measure of efficiency. The counter of labels scanned for the bi-directional algorithm is increased by one for each iteration of the forward search, and by 0.5 for each iteration of one of the two backward searches.

The algorithms considered for comparison are: the unidirectional search using TdMARTINS, the bi-directional search of BiTdMARTINS, the K-approximate BiTdMARTINS for different values of K, and the naive algorithm for the min-max relative regret problem. As additional reference, the table also shows information on the computation of a quickest path in $I_1$ using the time-dependent Dijkstra’s algorithm.

The naive algorithm enumerates all $s$-$t$ paths alternatively for $I_1$ and $I_2$ until an optimum path is found; it is implemented as a straightforward time-dependent generalization of an algorithm by Hershberger et al. [69] for the computation of the $k$-shortest paths. The row corresponding to this algorithm in Table 2.2 does not show the number of labels.
Table 2.3: Relative regret, run-time, and scanned labels for 3 instances.

<table>
<thead>
<tr>
<th>Max Rel Regret</th>
<th>Run-time (ms)</th>
<th>Labels Phase 1</th>
<th>Labels Phase 2</th>
<th>Labels Phase 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uni-dir</td>
<td>1.0328</td>
<td>954,267</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Bi-dir</td>
<td>1.0328</td>
<td>487,993</td>
<td>210,374</td>
<td>212,609</td>
</tr>
<tr>
<td>K=1.2</td>
<td>1.0861</td>
<td>190,960</td>
<td>210,374</td>
<td>96,108</td>
</tr>
<tr>
<td>K=1.4</td>
<td>1.1013</td>
<td>53,154</td>
<td>210,374</td>
<td>23,114</td>
</tr>
<tr>
<td>K=1.6</td>
<td>1.1068</td>
<td>6,180</td>
<td>210,374</td>
<td>4,209</td>
</tr>
<tr>
<td>K=1.8</td>
<td>1.1095</td>
<td>1,670</td>
<td>210,374</td>
<td>1,159</td>
</tr>
<tr>
<td>K=2.0</td>
<td>1.1095</td>
<td>975</td>
<td>210,374</td>
<td>197</td>
</tr>
</tbody>
</table>

scanned. Instead, we display the average number of iterations before finding a path in the intersection. Since each iteration consists of several (a number linear in \( n \)) repetitions of the time-dependent Dijkstra’s algorithm, we can get an idea on the number of labels scanned from the first two rows of the table.

The improvements of the bi-directional search over the uni-directional are considerable both for the run-time and for the number of scanned labels. The efficiency further increases if we allow an approximation factor \( K \) greater than 1. It appears however that there is a limit to the speed-up that can be obtained via approximation. The reason for this limit is that large values of \( K \) greatly reduce the amount of time spent by the algorithm in phases 2 and 3, but do not decrease the time in phase 1. In particular, for some value of \( K \), say \( K^* \), the algorithm spends no time at all in phase 2 because the termination condition is met as soon as the phase begins. All values of \( K \) greater than \( K^* \) will therefore result in similar run-time and number of scanned labels.

Results for 3 instances. Table 2.3 shows experimental results for the case of 3 instances (the third day being Thursday). Since in this case the operations of extract minimum, insertion and dominance checking necessary to implement Martins’ algorithm cannot be implemented efficiently, all algorithms are as a result much slower than in the previous case. The experimental evaluation is therefore not as thorough, and only 240 pairs of \( s,t \) vertices were considered.

We can see that the speed-up of the bi-directional search is still considerable, and an even more remarkable speed-up can be obtained via approximation. By setting an approximation factor of \( K = 2.0 \), compu-
tations that in the exact case require in average 25 minutes to terminate can be performed in less than one second.

### Single Backward Search

In our input data there is a strong correlation between the weights of a path in the different instances, since they represent travel times in very similar time periods. However, this correlation is not explicitly exploited by our algorithm. One might ask for a way to improve the efficiency of BiTD-Martins by considering this feature more directly. For example, for the case of 2 instances we might get some improvement by replacing the two backward searches with a single one that, for each backward edge $e \in \vec{E}$, considers weights of the kind

$$\vec{w}(e) = \min_{i \in \{1,2\}} \{ \vec{w}_i(e) \}.$$  

The correctness of this algorithm and its approximated variants follows trivially from that of BiTD-Martins. The benefits of this new algorithm over the original one are however not trivial to estimate. On the one hand, if

$$\max_{e \in \vec{E}} \{|\vec{w}_1(e) - \vec{w}_2(e)|\}$$

is small, the modified backward search will settle almost the same vertices as before, with almost the same values, at the price of one execution of Dijkstra’s algorithm instead of two. On the other hand, the lower bounds on the distances computed in the reverse graph are less
accurate. As a result, the number of labels scanned by the modified algorithm is larger. The benefit of the modified algorithm is, in loose terms, inversely proportional to eq. (2.8).

Figure 2.4 shows a plot of the average run-time and the number of labels settled by the original bi-directional algorithm (UNREL) and the modified one (REL) for different values $K$ of approximation and for the same 10,000 $s$-$t$ pairs. We can see that UNREL is faster and it also settles less labels than REL. However, the difference between the two is very small and further decreases for increasing values of $K$ until the point where the performance of the two algorithms is almost equal. It is an interesting open question to identify cases where the benefit of a single backward search takes over both the run-time and the number of labels scanned.
Chapter 3

Sampling Paths According to Their Length

Sampling (or generating) a “random” object from a large set of combinatorial objects is a fundamental problem arising in Statistical Physics, Mathematics, and Computer Science. In the robust routing application described in Chapter 2, we for example require sampling of a path from the intersection of two approximation sets. Because the number of objects to sample is typically huge (i.e., exponential in the size of the input), direct enumeration is unfeasible. An efficient sampling procedure is thus an important tool for studying statistical properties of “typical” instances. For many problems, sampling with uniform distribution and counting the number of objects are two computationally equivalent tasks [72], which in most cases are \#P-hard. This includes counting simple paths [126], even when restricting to planar graphs [116]. For these cases, sampling is instead considered according to distributions that are “close” to the desired one [124]. A most relevant technique for the design of this kind of sampling procedures is the Markov chain Monte Carlo method [25].

The Markov chain Monte Carlo method involves the design of a Markov chain whose states are the objects we wish to sample, and whose stationary distribution is the probability we want to use to sample them. The sampling procedure is a “random walk” on the Markov chain for a fixed number of steps, until we are certain that the probability of sampling one object is (approximately) the stationary distribution
of the chain. A most critical part of this method is proving that the Markov chain is rapidly mixing, i.e., the number of steps needed to reach the stationary distribution is polynomially bounded by the size of the input.

While rapidly mixing Markov chains are known for several hard problems, like graph coloring \([45, 46, 47, 67]\), knapsack \([104]\), perfect matchings \([71]\), independent sets \([20, 48, 97]\), there is essentially no positive result for the case of simple \(s-t\) paths on general graphs. The only Markov chain proposed for this setting is by Roberts and Kroese \([119]\), but that is however not rapidly mixing. Most of the positive results consider restricted paths over a lattice structure. In the simplest instance of these restrictions we have paths using only downward and rightward edges of the two-dimensional grid. This case has been analyzed by Luby et al. \([96]\) also for multiple source-destination paths, where the Markov chain can “get stuck” because all paths must be disjoint and thus some non-local moves are introduced. If we further impose the paths to stay above the main diagonal of a square grid, the so-called staircase walks, the number of such paths is given by the famous Catalan numbers \([43]\). Martin and Randall \([98]\) consider a Markov chain for sampling such paths where the weight of a path is the number of times it hits the diagonal. In the sampler by Greenberg et al. \([61]\), the weight of a path is instead the number of faces below it (the authors also consider the more general case of higher dimensional lattices). Finally, Randall and Sinclair \([117]\) consider the case where only one end of the path is fixed, and provide an efficient sampler for all such paths of a given length in the infinite \(d\)-dimensional lattice.

We consider the task of sampling simple paths between two given vertices in planar graphs according to a fixed probability distribution. Since in several applications it is natural to ask for paths of some fixed length, we thus consider the weighted version of the sampling problem in which the distribution depends on the length of the paths. We study a natural Markov chain in which each state corresponds to a simple \(s-t\) path in a given undirected, unweighted planar graph and a current path is modified according to a simple local rerouting operation. Roughly speaking, rerouting operations resulting in longer paths are “accepted” with small probability, while those resulting in shorter paths are always accepted. We show that the chain always converges to the Gibbs distribution on the paths weighted according to their lengths. In other words, the probability of sampling a specific path \(x\) depends only on its length \(|x|\) (i.e., the number of its edges) and it is of the
form
\[ \pi(x) \propto \lambda^{|x|}, \]  
\hspace{1cm} (3.1)

where \( \lambda > 0 \) is a parameter that can be used to “tune” the chain. Setting \( \lambda = 1 \) yields a uniform sampler over all \( s-t \) paths, while smaller/larger values provide samples biased towards shorter/longer paths.

Despite this Markov chain being not rapidly mixing in general planar graphs, we obtain an efficient sampler for the following setting, depicted in Figure 3.1. The paths from \( s \) to \( t \) are monotone in the vertical direction and the graph is any sub-grid of the 2D lattice without internal holes. The new Markov chain is a restriction of the original one maintaining the “vertical-monotonicity” of the paths. Our main technical contribution is a rigorous proof that this chain is rapidly mixing for all \( \lambda \in (0, 1] \). In the proof we combine the technique of path coupling without contraction [19] with the idea of modifying the chain by making some transitions “more lazy”. Note that in this restricted setting, an efficient sampler can also be obtained using dynamic programming (see Section 3.4). However, our interest lies in the analysis of the mixing time of the proposed Markov chain, which is a variant of a well-known “mountain/valley” chain [98].

We show that our results are tight in the following sense. First, the original “unrestricted” Markov chain is not rapidly mixing for \( \lambda = 1 \) in some planar graphs. This is true even for sub-graphs of the 2D lattice, and thus for the chain sampling all paths in a grid, without restricting to vertical-monotone ones. Both for the restricted and the unrestricted chains, we show that the mixing time is exponential in the number of vertices for every \( \lambda > 1 \). The latter result is in part expected because determining if a planar graph has an Hamiltonian path is \( \text{NP} \)-hard (and for sufficiently large \( \lambda \) a sampler can be used to solve this problem).
However, our negative results on the Markov chain are stronger in the sense that the chain remains slowly mixing even for very simple graphs where Hamiltonian paths can be computed efficiently. Thus, these results give a certain indication of the limitation of “local” chains. As for the case \( \lambda = 1 \), the existence of an efficient sampler for planar graphs remains an interesting open problem.

**Planar graphs.** Given a planar graph \( G = (V, E) \), we use \( n \) and \( m \) to denote respectively the number of vertices \( |V| \) and of undirected edges \( |E| \). By planarity, the vertices in \( V \) can be drawn as points in the plane in a way such that the edges in \( E \) are non-crossing curves; we denote such a drawing as a *plane embedding* of \( G \). In a plane embedding, a maximal region of the plane containing no points of the embedding is called a *face*; the unbounded region not enclosed by any edge is called the *outer face*. We use \( f \) to denote the overall number of faces, including the outer face. According to Euler’s formula, the number of faces satisfies \( f = m - n + 2 \). Note that in a planar graph \( f \in O(n) \). Namely, \( f \leq 2n - 4 \) with equality achieved by triangulated graphs, i.e., planar graphs in which every face is a triangle. Without loss of generality, we assume the graph to be 2-connected (we can otherwise easily reduce to this case).

**Markov chains and mixing time.** We consider Markov chains \( \mathcal{M} \) whose state space \( \Omega \) is finite. In our application, \( \Omega \) is the set of all simple \( s-t \) paths in a given planar graph. The matrix \( P \in [0, 1]^{\Omega \times \Omega} \) defines the transitions probabilities of \( \mathcal{M} \). We use \( P^t(x, y) \) to denote the probability that the chain moves in \( t \) steps from state \( x \) to state \( y \), where \( P^t \) is the \( t \)th power of \( P \).

A Markov chain is *irreducible* if, for all \( x, y \in \Omega \), there exists \( t \in \mathbb{N} \) such that \( P^t(x, y) > 0 \). In other words, every state can be reached with non-zero probability regardless of the starting state. The chain is *aperiodic* if, for all \( x \in \Omega \), \( \gcd\{t \in \mathbb{N} \mid P^t(x, x) > 0\} = 1 \). An aperiodic and irreducible Markov chain is called *ergodic*. It is well known \[85\] that an ergodic Markov chain converges to its stationary distribution \( \pi \), a unique vector \( \pi \in [0, 1]^{\Omega} \) such that \( \pi P = \pi \) and, for all \( x, y \in \Omega \), it holds

\[
\lim_{t \to \infty} P^t(x, y) = \pi(y).
\]

The *mixing time* of a Markov chain is the time needed for the distribution
P^t(x, \cdot) to get “sufficiently close” to the stationary distribution for any starting state x. Formally, the mixing time is defined as
\[ t_{mix}(\epsilon) := \min_{t \in \mathbb{N}} \max_{x \in \Omega} \{||P^t(x, \cdot) - \pi||_{TV} \leq \epsilon\}, \]
where \( ||P^t(x, \cdot) - \pi||_{TV} = \frac{1}{2} \sum_{y \in \Omega} |P^t(x, y) - \pi(y)| \) is the total variation distance. It is customary to define \( t_{mix} = t_{mix}(1/4) \) since it holds \( t_{mix}(\epsilon) \leq \lceil \log_2(1/\epsilon) \rceil t_{mix} \) [85]. A Markov chain is rapidly mixing if \( t_{mix}(\epsilon) \) is bounded from above by a polynomial in \( \log(|\Omega|) \) and in \( \log(1/\epsilon) \).

A rapidly mixing Markov chain can be used to efficiently sample elements from \( \Omega \) with probability arbitrarily close to \( \pi \). Simply simulate a random walk on the chain from an arbitrary initial state \( x \) for \( t = t_{mix}(\epsilon) \) time steps and return the state of the chain at time \( t \). The probability \( P^t(x, y) \) of the returned state \( y \) is approximately \( \pi(y) \).

### 3.1 A Markov Chain for Planar Graphs

We now define a Markov chain \( \mathcal{M}_{paths} \) whose state space \( \Omega \) is the set of all simple s-t paths of a given planar graph. The transitions of \( \mathcal{M}_{paths} \) are defined by rerouting a current path along one of its adjacent faces (see Figure 3.2).

**Definition 3.1.** Let \( x \) be an s-t path and \( f \) be a face adjacent to at least one edge of \( x \). We say that \( x \) can be rerouted along \( f \) if the edges in \( x \) and \( f \) form a single sub-path of \( x \) of length at least one, and the path \( y \) obtained by replacing all edges common to \( x \) and \( f \) with the edges in \( f \) that do not belong to \( x \) is simple. In this case, the rerouting operation consists of replacing \( x \) with \( y \).

Note that we forbid rerouting operations that reduce the length of the current path by introducing a cycle and short-cutting it afterwards. The reason is that these operations are not “reversible” and would prevent
us from using well-established methods to determine the stationary distribution\(^1\). Given a parameter \(\lambda > 0\), the transitions of \(\mathcal{M}_{\text{paths}}\) from a state \(x\) are defined according to the following rule:

1. With probability \(\frac{1}{2}\) do nothing. Otherwise,
2. Select a face \(f\) uniformly at random. If \(x\) cannot be rerouted along \(f\) then do nothing. Otherwise,
3. Move to the path \(y\) obtained by rerouting \(x\) along \(f\) with probability

\[
A(x, y) = \min \left\{ 1, \frac{\lambda |y|}{\lambda |x|} \right\},
\]

and do nothing with the remaining probability \(1 - A(x, y)\).

For \(\lambda < 1\), rerouting operations increasing the length of the current path by \(\ell\) are accepted with probability \(\lambda^\ell\), while those reducing it are always accepted. The converse happens for \(\lambda > 1\).

**Remark.** We stress that a planar graph can have different embeddings and the transitions of \(\mathcal{M}_{\text{paths}}\) depend on the particular given one. In the next section we show that the Markov chain is ergodic for any embedding.

**Remark.** For the sake of simplicity we forbid rerouting along the outer face. Doing so would not change the presented results significantly except in making the proofs less readable.

### 3.2 Analysis of \(\mathcal{M}_{\text{paths}}\)

**Ergodicity.** For all states \(x \in \Omega\) it holds that \(P(x, x) \geq 1/2\), and thus \(\mathcal{M}_{\text{paths}}\) is aperiodic. To show ergodicity, it then suffices to prove that any two states \(x, y \in \Omega\) are connected by a path with non-zero probability. For this purpose, we introduce the following notion of “distance” between \(s-t\) paths.

**Definition 3.2.** Given two \(s-t\) paths \(x\) and \(y\), a maximal sub-path common to \(x\) and \(y\) is an ordered sequence of vertices appearing in both paths and not contained in a longer sequence with the same property. We use \(\Delta_{xy}\) to denote the number of maximal sub-paths common to \(x\) and \(y\).
Note that the definition also allows “degenerate” sub-paths of just one vertex. For instance, if \( x \) and \( y \) have only the starting and the ending vertices in common, then \( \Delta_{xy} = 2 \). Figure 3.3 shows examples of paths with different values of \( \Delta_{xy} \).

**Lemma 3.3.** For any plane embedding of a graph \( G \) the following holds. If \( x \) and \( y \) are two s-t paths such that \( \Delta_{xy} = 2 \), then they are connected in \( \mathcal{M}_{paths} \) by a path of length at most \( f \), where \( f \) is the number of faces of \( G \).

**Proof.** Consider the subgraph of \( G \) induced by the edges of \( x \) and \( y \), and the drawing of this subgraph on the plane. Since \( \Delta_{xy} = 2 \), the plane is divided in exactly two regions, one of which is bounded by the edges of \( x \) and \( y \) and the other one is unbounded. The bounded region contains \( \delta_{xy} \) faces of \( G \), with \( 1 \leq \delta_{xy} \leq f \). We prove by induction on \( \delta_{xy} \) that in \( \mathcal{M}_{paths} \) there is a path between \( x \) and \( y \) of length at most \( \delta_{xy} \).

The base case for \( \delta_{xy} = 1 \) is trivial, because \( y \) can be obtained by rerouting \( x \) along exactly one face, the one contained in the bounded region. The path in \( \mathcal{M}_{paths} \) is then the transition from \( x \) to \( y \).

We now assume the claim to hold for every \( \delta_{xy} \leq k - 1 \) and prove it for \( \delta_{xy} = k \). Consider a face \( f \) contained in the bounded region having at least one edge which belongs to \( x \) but not to \( y \). We distinguish among the following cases, depicted by Figure 3.4:

1. **Face \( f \) does not contain a vertex from \( y \).** We define a path \( x' \) as follows. Let \( u \) and \( v \) be respectively the first and last vertex of \( f \) that we encounter when moving from \( s \) to \( t \) along \( x \). Let \( z \) be the sub-path of \( f \) connecting \( u \) to \( v \) and not containing any intermediate vertex from \( x \). Then \( x' \) is the concatenation of the sub-path of \( x \) from \( s \) to \( u \), the path \( z \), and the sub-path of \( x \) from \( v \) to \( t \). Note that \( \Delta_{xx'} = 2 \), since \( x \) and \( x' \) are identical from \( s \) to \( u \) and from \( v \) to

---

\[ ^1 \text{The analysis of non-reversible Markov chains is in general rather difficult and it is considered an interesting problem also for simple chains [38].} \]
Chapter 3. Sampling Paths According to Their Length

Figure 3.4: The case distinction of the proof of Lemma 3.3.

\( t \), and they have no common vertices between \( u \) and \( v \). Similarly, \( \Delta_{yx'} = 2 \) since \( z \) has no vertices from \( y \).

We next observe that \( \delta_{xx'} + \delta_{x'y} = \delta_{xy} \), because the sub-path \( z \) partitions the faces in the bounded region determined by \( x \) and \( y \). Note also that the partition is proper because the original bounded region contains \( \delta_{xy} \geq 2 \) faces, and therefore \( \delta_{xx'} \leq k - 1 \) and \( \delta_{x'y} \leq k - 1 \). We can thus apply the inductive hypothesis to find a path in \( M_{\text{paths}} \) from \( x \) to \( x' \) of length \( \delta_{xx'} \) and one from \( x' \) to \( y \) of length \( \delta_{x'y} \). The concatenation of the two yields a path in \( M_{\text{paths}} \) from \( x \) to \( y \) of length \( \delta_{xx'} + \delta_{x'y} = \delta_{xy} \).

2. Face \( f \) contains some vertex from \( y \). Let \( u \) and \( v \) be the first vertex of \( f \) that we encounter by following respectively \( x \) and \( y \) along the non-common sub-paths. Let \( z \) be the sub-path of \( f \) going from \( u \) to \( v \) without intermediate vertices from \( x \) and \( y \). Then \( x' \) is the concatenation of the sub-path of \( x \) from \( s \) to \( u \), the path \( z \), and the sub-path of \( y \) from \( v \) to \( t \). Note that \( \Delta_{xx'} = 2 \) and that \( \Delta_{yx'} = 2 \).

The same argument of the previous case implies \( \delta_{xx'} + \delta_{x'y} = \delta_{xy} \) and \( \delta_{xx'} \leq k - 1 \) and \( \delta_{x'y} \leq k - 1 \). Thus, by inductive hypothesis, we can find a path in \( M_{\text{paths}} \) from \( x \) to \( y \) of length \( \delta_{xx'} + \delta_{x'y} = \delta_{xy} \).

The statement follows from the fact that \( \delta_{xy} \leq f \). \( \Box \)

One might be tempted to define the distance between two paths as the number of faces inside the bounded region formed by the two. However, the presence of “nested” regions as in Figure 3.5 makes this definition incorrect. In the figure, transforming \( x \) in \( y \) using only rerouting operations can only be done by rerouting \( x \) along the same face multiple times. The next lemma allows us to reduce to the simpler case of a single bounded region.

**Lemma 3.4.** For any graph and any two s-t paths \( x \) and \( y \) such that \( \Delta_{xy} > 2 \), there exists an s-t path \( x' \) such that \( \Delta_{xx'} = 2 \) and \( \Delta_{x'y} = \Delta_{xy} - 1 \).
Proof. We construct $x'$ as in Figure 3.5. Starting from $s$, let $a$ be the last vertex for which $x$ and $y$ are identical. From $a$, we follow $y$ until we reach a vertex $b$ belonging to both $x$ and $y$. From $b$ we follow again $x$ until $t$. Observe that the resulting path $x'$ is simple since in between $a$ and $b$ there are no vertices belonging to $x$. By construction, $x$ and $x'$ are identical from $s$ to $a$ and from $b$ to $t$, while they differ between $a$ and $b$. Therefore, it holds that $\Delta_{xx'} = 2$.

We now show that $\Delta_{x'y} \leq \Delta_{xy} - 1$. Consider the set of $\Delta_{xy}$ maximal sub-paths common to $x$ and $y$. One of such sub-paths contains $a$, while another distinct one contains $b$. If we start at $b$ and follow $x$ towards $t$ we encounter at most $\Delta_{xy} - 2$ such sub-paths. Therefore, the number of maximal sub-paths common to $x'$ and $y$ between $b$ and $t$ is at most $\Delta_{xy} - 2$. Moreover, observe that $x'$ and $y$ are identical between $s$ and $b$ and therefore here we have only one maximal sub-path common to $x'$ and $y$. In total, there are at most $1 + \Delta_{xy} - 2$ maximal sub-paths common to $x'$ and $y$, that is, $\Delta_{x'y} \leq \Delta_{xy} - 1$. 

**Theorem 3.5.** For any plane embedding of a graph and any pair of vertices $s$ and $t$, the Markov chain $M_{paths}$ is ergodic with diameter at most $2n^2$.

Proof. By Lemmas 3.3 and 3.4, there exists a path in $M_{paths}$ from $x \in \Omega$ to $y \in \Omega$ of length at most $\Delta_{xy} \cdot f$. Since in any planar graph $f \leq 2n - 4$ and $\Delta_{xy} < n$, the distance between $x$ and $y$ is smaller than $2n^2$. 

**Stationary distribution.** We characterize the stationary distribution of the chain $M_{paths}$ by showing that, for some distribution $\pi$ and for all $x, y \in \Omega$, it holds

$$\pi(x) \cdot P(x, y) = \pi(y) \cdot P(y, x)$$

It is well-known that $\pi$ is then the stationary distribution of $M_{paths}$; this property is known as the detailed balance condition [85, Proposition 1.19].
Theorem 3.6. For any planar graph and any two vertices \( s \) and \( t \), the stationary distribution of the Markov chain \( M_{paths} \) is

\[
\pi(x) = \frac{\lambda^{|x|}}{Z(\lambda)},
\]

where

\[
Z(\lambda) = \sum_{z \in \Omega} \lambda^{|z|}.
\]

Proof. We show that \( \pi \) satisfies the detailed balance condition. Given \( x, y \in \Omega \) with \( P(x, y) > 0 \), assume without loss of generality that \( A(y, x) = 1 \) and observe that

\[
P(x, y) = \frac{1}{2f} \cdot \frac{\lambda^{|y|}}{\lambda^{|x|}}, \quad \text{and} \quad P(y, x) = \frac{1}{2f}.
\]

Therefore

\[
\pi(x) \cdot P(x, y) = \frac{\lambda^{|x|}}{Z(\lambda)} \cdot \frac{1}{2f} \cdot \frac{\lambda^{|y|}}{\lambda^{|x|}} = \pi(y) \cdot P(y, x).
\]

The parameter \( \lambda \) can be used to tune the stationary distribution of \( M_{paths} \). For example, by setting \( \lambda = 1 \) the stationary distribution is the uniform distribution over all simple \( s-t \) paths. We conclude this section by providing some negative results concerning the mixing time of \( M_{paths} \).

Theorem 3.7. For \( \lambda = 1 \), the Markov chain \( M_{paths} \) is not rapidly mixing.
Proof. We apply the well-known bottleneck theorem [85] which says that, for any subset of states $R \subset \Omega$ such that $\pi(R) \leq 1/2$, the following bound on the mixing time of the chain holds:

$$t_{\text{mix}} \geq \frac{\pi(R)}{4Q(R, \bar{R})},$$

where $Q(R, \bar{R}) = \sum_{x \in R, y \in \bar{R}} \pi(x)P(x, y)$.

Let $G$ be the planar graph obtained by combining two copies of a planar graph $H$ as follows. The two copies share only a single edge connecting $s$ and $t$, and all other vertices and edges in $H$ are duplicated (see Figure 3.6a). The set $R$ consists of the $s$-$t$ paths of $G$ using only edges in one of the two copies, say the upper one. Note that $R$ contains the common single-edge path $x^* = (s, t)$, that this is the only path in $R$ with transitions to some $y \in \bar{R}$, and that there are at most two transitions from $x^*$ to some other state (each edge is adjacent to at most two faces). Therefore

$$Q(R, \bar{R}) = \sum_{x \in R, y \in \bar{R}} \pi(x)P(x, y) \leq 2\pi(x^*) = 2/|\Omega|.$$

In order to apply the bottleneck theorem we need $\pi(R) \leq 1/2$. This can be easily achieved by adding two more paths to the bottom copy of $H$ (recall that $R$ consists of all paths in the upper copy of $H$). We then get

$$t_{\text{mix}} \geq \frac{\pi(R)}{4Q(R, \bar{R})} \geq \frac{|R|/|\Omega|}{8/|\Omega|} = \frac{|R|}{8}.$$

\[\square\]

Corollary 3.8. There exists an infinite family of planar graphs such that the mixing time of $M_{\text{paths}}$ satisfies $t_{\text{mix}} \in \Omega(\lambda^{n/2})$ for all $\lambda > 1$.

Proof. Consider the graph in Figure 3.6b consisting of a single row of $2k$ faces and $s$ and $t$ located in the middle as shown. Observe that

$$Z(\lambda) = \lambda + 2(\lambda^3 + \lambda^5 + \cdots + \lambda^{2k+1}).$$

We consider the bottleneck ratio of the set $R$ of all $s$-$t$ paths on the right part of the graph and of length larger than 1. Thus

$$\pi(R) = \frac{\lambda^3 + \lambda^5 + \cdots + \lambda^{2k+1}}{Z(\lambda)} \leq \frac{1}{2}.$$
Chapter 3. Sampling Paths According to Their Length

Figure 3.7: Transitions of $\mathcal{M}_{\text{mon}}$; $p$ is the probability of selecting a face.

Note that the only $s$-$t$ path in $R$ which has a non-zero transition probability to some $y^* \in \bar{R}$ is the path $x^*$ of length 3, and the only one such $y^* \in \bar{R}$ is the path $(s,t)$ of length 1. Therefore, we have

$$Q(R, \bar{R}) = \pi(x^*)P(x^*, y^*) = \frac{\lambda^3}{Z(\lambda)} \cdot \frac{1}{2f} \cdot \frac{\lambda}{\lambda^3}.$$ 

Thus, the bottleneck theorem implies (we also use that $f \geq 2$)

$$t_{\text{mix}} \geq \frac{\pi(R)}{4Q(R, \bar{R})} = \frac{\lambda^3 + \lambda^5 + \cdots + \lambda^{2k+1}}{4\lambda/2f} > \lambda^{2k}.$$ 

Since the graph has $n = 2(2k + 1) = 4k + 2$ vertices, we get $t_{\text{mix}} \in \Omega(\lambda^{n/2})$.

We note that Theorem 3.7 holds also for graphs that have a very simple structure, like two square grids sharing only the edge $(s,t)$ and outerplanar graphs.

### 3.3 A Rapidly Mixing Chain

We now present the rapidly mixing Markov chain $\mathcal{M}_{\text{mon}}$, which is a natural modification of $\mathcal{M}_{\text{paths}}$ for the case where the graph is a sub-graph of the two-dimensional lattice (grid) with no holes. That is, every face is either a cell of the grid or it is the outer face. The chain $\mathcal{M}_{\text{mon}}$ samples paths that are vertical-monotone, that is, that are only monotone in the vertical direction (if we follow the path from $s$ to $t$, it never goes up). We thus assume that $s$ lies above or at the same $y$-coordinate of $t$. Though it is straightforward to generate such paths uniformly at random, our goal is a weighted sampler with probability biased towards shorter paths according to the parameter $\lambda$, i.e., the distribution of eq. (3.1).

The Markov chain is a modification of $\mathcal{M}_{\text{paths}}$ in which some transitions are disallowed and others are “more lazy” (see Figure 3.7). In particular,
the chain does not allow to replace a horizontal edge with three edges, and transitions swapping two consecutive edges of a face are only performed with probability
\[ \gamma := \frac{1 + \delta}{2}, \text{ where } \delta = \lambda^2 \text{ and } \lambda \in (0, 1). \]

This choice of \( \gamma \) will be useful for the analysis of the mixing time. Note that we restrict to the case \( \lambda \leq 1 \), because the lower bound for \( \lambda > 1 \) of Corollary 3.8 holds also for \( \mathcal{M}_{\text{mon}} \). Note further that \( \mathcal{M}_{\text{mon}} \) is ergodic with diameter \( \leq 2f \), and its stationary distribution is the same as for \( \mathcal{M}_{\text{paths}} \).

To bound the mixing time we use the method of path coupling without contraction [19]. Given a Markov chain \( \mathcal{M} \) with state space \( \Omega \), a path coupling is a stochastic process \((X, Y)\) on \( \Omega \times \Omega \) with distributions
\[ P_{x,y}[X = x', Y = y'] \]
for all \( x, y \in \Omega \), such that
\[ P_{x,y}[X = x'] = P(x, x') \text{ for all } x' \in \Omega, \]
\[ P_{x,y}[Y = y'] = P(y, y') \text{ for all } y' \in \Omega. \]

We use \( \rho \) to denote the shortest-path distance in \( \mathcal{M} \); that is, \( \rho(x, y) \) indicates the minimum number of transitions to go from \( x \) to \( y \).

**Lemma 3.9** (Theorem 2 in [19]). Let \((X, Y)\) be a path coupling for a Markov chain \( \mathcal{M} \) such that, for all \( x, y \) with \( P(x, y) > 0 \), it holds
\[ \mathbb{E}_{x,y}[\rho(X, Y)] \leq 1. \tag{3.2} \]

Then, the Markov chain \( \mathcal{M}^* \) with transition matrix \( P^* = (P + p_{\text{min}}I)/(1 + p_{\text{min}}) \) has mixing time \( t_{\text{mix}}^* \in O \left( \frac{D^2}{p_{\text{min}}} \right) \), where \( p_{\text{min}} \) and \( D \) denote respectively the smallest non-zero transition probability and the diameter of \( \mathcal{M} \).

Note that \( \mathcal{M}^* \) is the Markov chain with transition probabilities
\[ P^*(x, y) = \begin{cases} \frac{P(x,x) + p_{\text{min}}}{1 + p_{\text{min}}} & \text{if } y = x, \\ \frac{P(x,y)}{1 + p_{\text{min}}} & \text{otherwise}. \end{cases} \]

It is easy to see that \( \mathcal{M}^* \) and \( \mathcal{M} \) have the same stationary distribution. We can thus run the chain \( \mathcal{M}^*_{\text{mon}} \) to efficiently sample vertical-monotone paths.
Path coupling for $\mathcal{M}_{\text{mon}}$. For the sake of clarity, for every face $f$ we define the following shorthand:

$$p_f(x) = P(x, x \oplus f),$$

where $x \oplus f$ denotes the path obtained by rerouting $x$ along $f$. We define a path coupling by specifying, for every pair $(x, y)$ such that $x$ and $y$ differ in one face $d$, the probabilities to move to $(x', y')$:

$$(x, y) \mapsto (x \oplus d, y) \quad \text{with prob. } p_d(x), \quad (3.3)$$

$$(x, y) \mapsto (x, y \oplus d) \quad \text{with prob. } p_d(y), \quad (3.4)$$

and for every other face $f \neq d$

$$(x, y) \mapsto (x \oplus f, y \oplus f) \quad \text{with prob. } \min\{p_f(x), p_f(y)\}, \quad (3.5)$$

$$(x, y) \mapsto (x \oplus f, y) \quad \text{with prob. } \max\{0, p_f(x) - p_f(y)\}, \quad (3.6)$$

$$(x, y) \mapsto (x, y \oplus f) \quad \text{with prob. } \max\{0, p_f(y) - p_f(x)\}. \quad (3.7)$$

Finally, with all remaining probability

$$(x, y) \mapsto (x, y). \quad (3.8)$$

It is easy to check that this is indeed a path coupling. The difficulty is in proving the condition necessary to apply Lemma 3.9. For this purpose, consider two paths $x, y \in \Omega$ such that $\rho(x, y) = 1$. Note that the transitions defining our path coupling correspond to the following values for $\rho(X, Y)$:

$$\rho(X, Y) = 0 \quad \text{for (3.3) and (3.4)},$$

$$\rho(X, Y) = 2 \quad \text{for (3.6) and (3.7)},$$

$$\rho(X, Y) = 1 \quad \text{for (3.5) and (3.8)}.$$

Therefore, it holds

$$\mathbb{E}_{x,y}[\rho(X, Y)] = 0 \cdot p_0 + 1 \cdot p_1 + 2 \cdot p_2 = 1 + p_2 - p_0,$$

where

$$p_0 = \mathbb{P}_{x,y}[\rho(X, Y) = 0] = p_d(x) + p_d(y),$$

$$p_2 = \mathbb{P}_{x,y}[\rho(X, Y) = 2] = \sum_{f \neq d} |p_f(x) - p_f(y)|,$$

$$p_1 = \mathbb{P}_{x,y}[\rho(X, Y) = 1] = 1 - p_2 - p_0.$$
Figure 3.8: The names of the faces around d.

Figure 3.9: The cases in the proof of Lemma 3.10.

Our goal is then to prove that \( p_2 \leq p_0 \) for all \( x, y \in \Omega \) with \( \rho(x, y) = 1 \). In order to bound \( p_2 \) it is enough to consider only the faces adjacent to d in which \( x \) and \( y \) differ. We name them according to Figure 3.8. The following lemma uses the monotonicity of the paths to prove that “corner faces” can be ignored.

**Lemma 3.10.** If \( f \in \{nw, ne, sw, se\} \) then \( |p_f(x) - p_f(y)| = 0 \).

**Proof.** Consider the case \( f = nw \) (the other cases are similar) and the intersection of this face with the subpath common to both \( x \) and \( y \), depicted in Figure 3.9. Observe that, if vertex \( d \) is in \( x \) and \( y \), then \( p_{nw}(x) = p_{nw}(y) \). Now consider the case where \( d \) belongs only to one of the two paths, say \( x \), and the edges of \( nw \) common to both \( x \) and \( y \). If the edge \((a, c)\) is common (Figure 3.9a), the monotonicity of the paths implies that also \((c, d)\) is common (Figure 3.9b), contradicting the hypothesis that \( d \) is not on \( y \). Thus, we have that only \((a, b)\) can be common (Figure 3.9c), but then the paths cannot be rerouted along \( nw \) and thus \( p_{nw}(x) = p_{nw}(y) = 0 \). \( \square \)

**Lemma 3.11.** The path coupling defined above satisfies eq. (3.2).

**Proof.** By Lemma 3.10, our goal is to show that

\[
p_2 = \sum_{f \in \{n, s, e, w\}} |p_f(x) - p_f(y)| \leq (1 + \delta)p = 2\gamma p = p_0.
\]
Figure 3.10: The cases where $x$ and $y$ have the same length.

Dividing all terms by $p$, this is equivalent to show the inequality

$$p_n + p_s + p_e + p_w \leq 1 + \delta = 2\gamma,$$

where

$$p_f = \frac{|p_f(x) - p_f(y)|}{p} \quad \text{for every face } f.$$

We distinguish several cases which we group according to Figure 3.10 and Figure 3.11. For Figure 3.10, it holds

- **Figure 3.10a:** $p_n = \gamma - \delta$, $p_s = 0$, $p_e = \gamma$, $p_w = \delta$
- **Figure 3.10b:** $p_n = \gamma - \delta$, $p_s = \gamma - \delta$, $p_e = \delta$, $p_w = \delta$
- **Figure 3.10c:** $p_n = 0$, $p_s = \gamma - \delta$, $p_e = \delta$, $p_w = \gamma$.

The remaining cases are similar to the previous ones as they differ in only one face. For instance, the only difference between the case in
Figure 3.10d and the one in Figure 3.10a is for face n. However, since $\gamma = \frac{1+\delta}{2}$, in the case of Figure 3.10d we have $p_n = 1 - \gamma = \gamma - \delta$, that is, the same as in Figure 3.10a.

For Figure 3.11, it holds

Figure 3.11a: $p_n = 0$  $p_s = 0$  $p_e = 1$  $p_w = \delta$
Figure 3.11b: $p_n = \gamma - \delta$  $p_s = 0$  $p_e = \gamma$  $p_w = \delta$
Figure 3.11c: $p_n = 0$  $p_s = \gamma - \delta$  $p_e = \gamma$  $p_w = \delta$
Figure 3.11d: $p_n = \gamma - \delta$  $p_s = \gamma - \delta$  $p_e = \delta$  $p_w = \delta$.

The remaining cases follow from the identity $1 - \gamma = \gamma - \delta$.

To conclude the proof, we note that the case where d is not surrounded by four faces, that is, d is on the border of the grid, is even more favorable than those above, because some faces among n, s, e, w will not be present.

Since the diameter of $M_{mon}$ is in $O(n)$ and the minimum non-zero
transition probability is $p_{\text{min}} = \frac{\lambda^2}{2f}$, we can establish the following bound.

**Theorem 3.12.** The mixing time of $\mathcal{M}_{\text{mon}}^*$ is in $O\left(n^3/\lambda^2\right)$ for all $\lambda \in (0,1]$.

### 3.4 A Dynamic Programming Sampler

For the sake of completeness, we describe a simple dynamic programming approach for sampling vertical-monotone paths. For this purpose, we assume to have available a polynomial-time procedure that returns a path selected uniformly from the subset of all paths of length $\ell$, and that we can efficiently compute the number $m_\ell$ of $s$-$t$ paths of length $\ell$. We can use this procedure to sample paths according to the Gibbs distribution (3.1) as follows:

1. Select a length $k$ with probability
   \[ s_k = \frac{m_k \lambda^k}{Z(\lambda)}, \]
   where $Z(\lambda) = \sum_{z \in \Omega} \lambda_{|z|} = \sum_\ell m_\ell \cdot \lambda^\ell$.

2. Select a path $x$ uniformly at random in the set of all $s$-$t$ paths of length exactly $k$.

To see that we are indeed sampling according to (3.1), observe that the probability that a path $x$ is selected is $s_k \cdot (1/m_k) = \lambda^k / Z(\lambda)$.

We can realize the above sampler using the following standard dynamic programming algorithm for counting all non-simple paths of length up to some value $L$. For vertical-monotone paths, this algorithm can be trivially modified to avoid non-simple paths. Since we consider unweighted graphs, it holds $L \leq n$. We compute a table $T(u, k)$ for all vertices $u$ and for all lengths $k$ such that $T(u, k)$ is the number of paths of length $k$ from $s$ to $u$. Initially $T$ is identical to 0 except for $T(s, 0) = 1$. Given the values for $k = 0, \ldots, \ell - 1$ we compute $T(u, \ell)$ as follows. For every $u$, consider all neighbors $v \in N_u$ and let

\[ T(u, \ell) = \sum_{v \in N_u} T(v, \ell - 1). \]  

(3.9)

In our application, the neighbors of $u$ are the vertices on the grid that are placed below, on left, and on the right of $u$. 
We use the table $T$ to pick a path $x$ uniformly at random among all $s$-$t$ paths of length $\ell$. Consider the entry $T(t, \ell)$ indicating the number of such paths. Starting from $u = t$, we select a neighbor $v$ of $u$ with probability

$$\eta_v = \frac{T(v, \ell - 1)}{T(u, \ell)},$$

and repeat this step from the selected vertex until we reach $s$. The resulting sequence of vertices is an $s$-$t$ path $x = \langle x_0, \ldots, x_{\ell - 1} \rangle$ of length $\ell$. To see that $x$ is chosen with probability $1/T(t, \ell)$, observe that the probability of selecting $x_{\ell - 1}, x_{\ell - 2}, \ldots, x_0$ is

$$\eta_{x_{\ell - 1}} \eta_{x_{\ell - 2}} \cdots \eta_{x_0} = \frac{T(x_{\ell - 1}, \ell - 1)}{T(x_{\ell}, \ell)} \cdot \frac{T(x_{\ell - 2}, \ell - 2)}{T(x_{\ell - 1}, \ell - 1)} \cdots \frac{T(x_0, 0)}{T(x_{\ell - 1}, \ell - 1)} = \frac{T(s, 0)}{T(t, \ell)} = \frac{1}{T(t, \ell)}.$$

To conclude the analysis we provide a simple upper bound on the running time. First, the table has size $n \times n$ and it can be computed in time $O(n^2)$, since each node has only a constant number of neighbors. For the same reason, the procedure for selecting a random $s$-$t$ path of length $\ell$ takes $O(\ell)$ steps, and thus the overall running time is in $O(n^2)$. 
The sources of uncertainty in geometric applications are various and ubiquitous. A typical example is given by the popular method for measuring locations, the Global Positioning System (GPS). A GPS receiver determines its global position according to measured distances from a set of satellites orbiting the Earth. These distances are however calculated within an error interval, and as a result the estimated position of the GPS receiver might not be completely accurate. Furthermore, trajectories are usually represented as a sequence of measured locations that are later interpolated to obtain a curve. In addition to the imprecision of the measurements of the single locations, we then have to deal with the imprecision introduced by the chosen interpolation. Finally, the representation of shapes is usually done using geometric graphs (for example, triangulations or polygons). If the locations of the vertices of these graphs are measured by an imprecise system, the resulting shape will therefore be imprecise. In scientific terminology we distinguish between the accuracy and the precision of a measurement system. We say that a system is accurate if the measures it produces are “close” to the real measurements, and we say that it is precise if repeated measurements of the same object agree. A system that is precise might very well be inaccurate and vice versa. In this thesis we consider systems that are accurate but imprecise and investigate the algorithmic problems that arise while working with these systems. We assume a worst-case perspective by adopting a model that has been popularized by Löffler [90] under the name of imprecise points. In Chapter 4 we present this model as well as highlight the main results. In Chapters 5 and 6 we apply the model respectively to the shortest path and spanning tree problem and investigate the related algorithmic questions.
Chapter 4

Models of Imprecision

With the imprecise points model we assume a worst-case perspective on the topic of data imprecision. We assume that the exact location of each input point is unknown, but that we do have a minimal idea of where in the plane (or space) it might be located. We represent this setting by replacing each point with a set of possible locations, an occurrence region for that point. The “real” point might be located anywhere in the corresponding region. This model generalizes straightforwardly to a more complex type of input like geometric graphs, i.e., a graph whose vertex set consists of points in the plane, by replacing each vertex with a corresponding occurrence region. However, for input consisting of lines (or curves), it is not clear how to apply the imprecise points model. Requirements of connectedness or convexity of the set of all possible lines are neither trivial nor uniquely defined. Given that the focus of this thesis lies in the algorithmic aspects of uncertainty rather than the modeling issues, we will not consider the topic of imprecise lines. In Section 4.1 we present the main results related to the imprecise points model. Alternative models are briefly illustrated in Section 4.2.

4.1 Imprecise Points

Without imprecision, the output of a geometric problem is unambiguous and well-defined. The same however cannot be said when the same problem is cast into the world of point imprecision. Consider for
example the computation of the weight of a minimum spanning tree of a point set. When the points are imprecise, a possible output could be the smallest weight that such a spanning tree can attain. However, in different applications we might instead be interested in the largest weight of such a tree. Formalizing concrete questions that can be asked in the setting of points imprecision is a non-trivial task. To coherently present the main results related to the imprecise points model, we categorize them according to the type of output required by the problem in its exact (i.e., precise) variant.

**Boolean output.** The output of a decision problem is a single Boolean value, a yes/no answer to the question of whether a given property is satisfied by the input data or not. Decision problems for imprecise points often represent very fundamental questions for the whole field of computational geometry and therefore have been considered independently from the context of data uncertainty. For example, given a set of regions in the plane, determining whether there exists a line passing through all regions is well-known as the *transversal* or *stabbing* problem. This problem and several variants of it have been studied extensively [49, 121]. Even though stabbing problems can be considered related to data uncertainty, their scope is clearly much more general.

Guibas et al. [63] were among the first to consider point imprecision explicitly by introducing the notion of \( \varepsilon \)-predicates, which are predicates defined on a point set that can become true if the position of each point is perturbed by at most \( \varepsilon \). The authors demonstrate how to implement these predicates for tests such as collinearity or orientation.

**Numeric output.** Typical numeric measures of a points set are its diameter, its width, or the radius of its smallest enclosing circle. If the input consists of imprecise points, we are typically interested instead in the smallest or largest value that such measures can attain. It is possible in this case to require the numeric output to be paired with a *realization*, that is, a corresponding set of points, each drawn from its own occurrence region, attaining that value.

Löffler and van Kreveld [95] study basic measures such as the diameter, the width, the closest pair, the area of the smallest enclosing box and the radius of the smallest enclosing circle for imprecise points in squares or disks in the plane, while Kruger [84] considers these measures for
Imprecise Points

balls in high dimensions. Löffler and van Kreveld [94] also consider the problems of minimizing or maximizing the area or the perimeter of the convex hull for imprecise points in squares or line segments, and provide polynomial-time algorithms for many different variants. The same authors [127] prove that the problem of maximizing the area of the convex hull for imprecise points in arbitrary (possibly intersecting) line segments is NP-hard, and provide linear-time approximation schemes.

A more involved measure on a point set is the cost of a minimum spanning tree, that can be defined as the weight, i.e., the sum of the lengths of its edges, or as the diameter, i.e., the length of its longest simple path. Dorrigiv et al. [42] show that the problem of minimizing the weight of a spanning tree for imprecise points in disjoint disks does not admit an FPTAS. In Section 6.1 we extend this result to imprecise points in axis-aligned segments. Concerning the minimization of the diameter, we provide in Section 6.2 polynomial-time algorithms for imprecise points in different kinds of disks with running times ranging between $O(n^5)$ and $O(n^9)$.

When the input is more complex, like an imprecise geometric graph, we can also be interested in numeric bounds on its possible realizations. Dror et al. [44] consider the problem of minimizing the perimeter of a graph representing a polygon in the plane, while Polishchuk and Mitchell [115] study the same problem in higher dimensions. This problem is closely related to the diameter of a graph, i.e., the length of its longest shortest path, and to its girth, i.e., the length of its shortest cycle. In order to provide bounds on these measures, we must however first be able to provide bounds on the shortest path between any pair of vertices of the geometric graph. We present our results on this topic in Chapter 5.

**Combinatorial output.** For some problems the output cannot be expressed with a primitive value like a Boolean or a number. This is for example the case for the computation of a certain region or subdivision of the plane, like the convex hull or the Voronoi diagram. Dealing with imprecise points, an interesting question in these cases is to compute the extreme boundaries of such a region or subdivision.

For convex hulls, the question translates to the computation of the union or the intersection of the convex hulls for all realizations of the imprecise points. Nagai and Tokura [106] consider this problem for
imprecise points in convex polygons, while Sember and Evans [123] study the union of the convex hulls for non-convex polygons.

For Voronoi diagrams we can look at the computation of the inner boundaries of each Voronoi region, i.e., the set of points that are in the region for every possible realization of the imprecise points. Khanban [80] considers this problem for imprecise points in rectangles, while Sember and Evans [122] propose an optimal algorithm for disks.

Pre-processing. In many applications we expect to be asked the same query several times (for example, point location queries). It is important in these cases to pre-process the input into a data structure that allow us to answer these queries more efficiently than we would without the data structure on hand. For imprecise points we can make a similar request; the goal now is to be “ready” for when the point set becomes precise. In particular, we want to compute a data structure that allows us, when a realization is given, to answer the underlying algorithmic question more efficiently than we would without the pre-processing phase.

This setting was first explored by Held and Mitchell [68], that proposed a simple and practical algorithm to pre-process a set of disks so that, when a point inside each disk is specified, a triangulation of the point set can be computed in linear time; Kreveld et al. [128] extend this result to disjoint polygons. If we additionally require the triangulation to be Delaunay, further results are known for imprecise points in disks or fat regions [26, 92]. Pre-processing techniques are also known for the computation of the convex hull for imprecise points in lines and segments [51], and for the onion decomposition in disks [91].

4.2 Alternative Models

The imprecise points model assumes a minimal perspective on the issue of uncertainty, where the only information we have about a point are the possible locations where it may lie. In many applications, more information about the input data might be available. In the following we review some of the methods to model these situations.
Probabilistic methods. An additional bit of information that may be provided together with the uncertain input are probability distributions. For example, every point of a point set might be given as a set of possible locations, each with a corresponding probability for the point to be actually located there. The model where each point is given as a finite set of $k$ possible locations is known as *indecisive points* [2, 73]. Note that we can permit a point to not be present at all in a realization by allowing the sum of the probability of each location to be smaller than 1. By further restricting $k$ to be equal to 1, we obtain the so-called *stochastic points* model [74, 75].

Dependencies. A hidden assumption in the previous models is that the imprecision of an input point is independent from the imprecision of the other points in the set. This assumption is reasonable if locations are obtained using independent measurements, but sometimes we might instead expect the location of the points to be correlated. It is our wish then to exploit this correlation in the computation of a solution. The *linear parametric geometric uncertainty model* [105] defines dependencies between points using a set of common parameters on which all points depend.

Update complexity. An imprecise measurement system might be capable of more precise measurements if we allow it the use of more resources (like time or energy). The goal in these cases is to compute the required geometric objects, or their costs, while minimizing the additional resources spent for the more precise measurements [24, 53].
Chapter 5

Shortest Paths

In this chapter we consider a set of simple polygons \( \mathcal{P} = \{P_1, \ldots, P_n\} \) in the plane and a graph \( G = (\mathcal{P}, E) \) connecting them as in Figure 5.1a. Placing a point inside each polygon turns \( G \) into a geometric graph \( G_p, p = \{p_1, \ldots, p_n\} \) where the weight of \( (P_i, P_j) \in E \) is the distance between \( p_i \) and \( p_j \). Given two polygons \( P_s, P_t \in \mathcal{P} \), we study the shortest path distance between the points in \( P_s \) and \( P_t \). In particular, we consider the task of placing a point inside each polygon such that the weight of a shortest path between the points in \( P_s \) and \( P_t \) is either the largest or the smallest among all possible placements. We refer to the problem in its minimization and maximization variants respectively as the Minimum and Maximum Shortest Path Problem for Imprecise Points (in short MIN-SPP and MAX-SPP). Figures 5.1b to 5.1d show examples of optimum placements for various metrics.

The shortest path problem has been considered in the setting of points imprecision mainly in a variant where the underlying graph is a path. This variant, generally known as the Touring Polygons Problem (in short TPP), is stated as follows. Given a sequence of simple polygons and two points \( s \) and \( t \), the problem asks for a shortest tour starting at \( s \) and ending at \( t \) while visiting all polygons in the given order. The term “tour” comes from the fact that typically (but not necessarily) \( s \) and \( t \) are the same point. The problem was introduced by Dror et al. [44], who showed it to be solvable in polynomial time if the polygons are convex and NP-hard in the case of non-convex overlapping polygons. Ahadi et al. [4] strengthen the latter result by proving that the TPP is NP-hard if
Figure 5.1: (a) An instance of SPP. (b)-(c) Optimum MIN-SPP and MAX-SPP placements for $L_2$ metric. (d) Optimum placement for $L_1$.

the non-convex polygons are disjoint, even if each polygon consists of two joint segments whose angles with the $x$-axis are in \( \{0, \pm \pi/4, \pi/2\} \). Polynomial time approximation schemes are further known for the non-convex case [44, 113]. We point out that the only variant of the TPP with non-convex polygons admitting a polynomial time algorithm is known for the case of rectilinear axis-aligned polygons under the $L_1$ metric [44]. For the same kind of polygons, the complexity of the TPP under metrics other than $L_1$ is open.

The SPP is a natural generalization of the TPP. Instead of an ordered sequence of polygons we are now given a directed graph connecting them; the presence of an edge indicates an “allowed traversal” from one polygon to the other. In general, the MIN-SPP is NP-hard for any metric $L_p$, $p \geq 1$, even for disjoint segments whose angles with the $x$-axis are in \( \{0, \pm \pi/4, \pi/2\} \). The proof is a straightforward generalization of the one by Ahadi et al. [4] of the TPP for non-convex disjoint polygons. We do not include the proof in this text, but the idea is to consider every polygon $P$ composed of two joint segments used in the reduction from Ahadi et al. and replace it with its two parts. We then introduce in the
5.1 Min Shortest Path

In the variant of the MIN-SPP we consider, the polygons are axis-aligned rectilinear. We use $\mathcal{R} = \{R_1, \ldots, R_n\}$ to denote the set of all polygons and $G = (\mathcal{R}, E)$ the underlying graph connecting them. The weight of an edge $(R_i, R_j) \in E$ in the geometric graph $G_p$ of a placement $p = \{p_1, \ldots, p_n\}$ is the $L_1$ distance $\|p_i - p_j\|_1$ (we will drop the subscript when it is clear from the context which metric we are referring to). In the following, we will make use of a structure called the Hanan grid [65], obtained from a given set of points $p$ by imposing horizontal and vertical lines through each point $p \in p$.

Our solutions relies on the existence of a minimum placement $p$ for the MIN-SPP with two particular properties. The first property is that...
the points in $p$ can be found on the intersections of the Hanan grid of the corners of $R$. The second property is that it further holds that each $p_i \in p$ lies on the boundary of the respective polygon $R_i$. Figure 5.2 shows an example of a minimum placement with these properties. We design an algorithm computing such a minimum placement in time $O(mnk^3)$, where $n = |R|$, $m = |E|$, and $k$ is the maximum number of corners of a polygon in $R$. The existence of this peculiar placement $p$ is proven in Lemma 5.2 using the following well-known fact on the $L_1$ metric.

**Fact 5.1.** Let the bounding box $B_{xy}$ of $x, y \in \mathbb{R}^2$ be the smallest axis-parallel rectangle containing $x$ and $y$. For every $x, y, z \in \mathbb{R}^2$, it holds

\[
z \in B_{xy} \iff \|x - y\| = \|x - z\| + \|z - y\|
\]

\[
z \notin B_{xy} \iff \|x - y\| < \|x - z\| + \|z - y\|.
\]

**Lemma 5.2.** There exists a minimum placement $p$ where every $p_i \in p$ lies on the boundary of $R_i$ and is a grid point of the Hanan grid induced by the corners of $R$.

**Proof.** Let $p$ be an optimum placement and $P$ be a shortest $s$-$t$ path in $G_p$. We show how to move the points in $p$ not satisfying the lemma to the Hanan grid and on the boundary of the respective polygons in a way such that the resulting placement is still optimum. A point in $p$ of a polygon not on (visited by) $P$ not satisfying the lemma can be trivially moved to an arbitrary corner of that polygon. Since the weight of $P$ in the resulting placement is the same as in $G_p$, the resulting placement is still optimum.

For the polygons on $P$ whose points in $p$ do not satisfy the lemma, we first show how to move the points on the boundary in a way such that the resulting placement is still optimum. Observe that $p_s$ trivially lies on the boundary of $R_s$; otherwise, we can obtain a better placement by moving $p_s$ closer to the point in the successor of $R_s$ on $P$, contradicting the optimality of $p$. The same argument holds by symmetry for $p_t$.

Let $R_j \notin \{R_s, R_t\}$ be a polygon on $P$ and consider $p_i, p_k \in p$, where $R_i$ and $R_k$ are respectively the predecessor and the successor of $R_j$ on $P$. Let $p_c$ be a point on the boundary of $R_j$ contained in the bounding box $B_{ij}$. By Fact 5.1 and the triangle inequality, it holds

\[
\|p_i - p_j\| + \|p_j - p_k\| = \|p_i - p_c\| + \|p_c - p_j\| + \|p_j - p_k\| \\
\geq \|p_i - p_c\| + \|p_c - p_k\|.
\]
Thus, moving $p_j$ to $p_c$ does not increase the weight of $P$. The resulting placement is still optimum, and $p_j$ now lies on the boundary of $R_j$. We can apply this operation to every point in the interior of its polygon.

To show how to move points in $p$ to Hanan grid points, we assume by the above each point of $p$ to lie on the boundary of its respective polygon, and that only points of polygons on $P$ may not be grid points. Let $p_j = (x_j, y_j) \in p$ be a point on the boundary of $R_j$ not on the Hanan grid. Since $R_j$ is axis-parallel, $p_j$ lies on a line of the grid. Thus, either $x_j$ is the $x$-coordinate of a grid point, or $y_j$ is the $y$-coordinate of a grid point. Without loss of generality, we consider only the latter case.

Let $x_l$ be the largest $x$-coordinate of a grid point lying to the left of $p_j$, and $x_r$ be the smallest $x$-coordinate of a grid point lying to the right of $p_j$. We define the vertical stripe of $p_j$ as the set $\{(x, y) \in \mathbb{R}^2 \mid x_l < x < x_r\}$. See Figure 5.3 for an example of a vertical stripe.

Consider a sequence $R_{i_r}, \ldots, R_k$ of consecutive polygons on $P$ of maximal length containing $R_j$ and such that every point in $p$ of a polygon in the sequence lies in the vertical stripe of $p_j$. Note that none of the points in the sequence is a grid point, but the $y$-coordinate of all such points are $y$-coordinates of grid points. We first consider the case where $R_i \neq R_s$ and $R_k \neq R_t$.

Let $R_{i'}$ and $R_k'$ be respectively the predecessor of $R_i$ and the successor of $R_k$ on $P$. If $p_{i'}$ lies to the left of the vertical stripe of $p_j$, we move every point $p_i, \ldots, p_k$ horizontally to the $x$-coordinate $x_l$. Otherwise, we
move them horizontally to $x_r$. Figure 5.3 illustrates such a movement. The weight difference of $P$ before and after moving the points is

$$
\sum_{(R_a,R_b) \in P'} \|p_a - p_b\| - \|p'_a - p'_b\|, \quad (5.1)
$$

where $P'$ is the sub-path of $P$ between $R_i'$ and $R_k'$, and $p_a'$ (resp. $p_b'$) is the new location of $p_a$ ($p_b$). Since points are only moved horizontally, their $y$-differences do not change. Thus, we can rewrite eq. (5.1) as

$$
\sum_{(R_a,R_b) \in P'} |x_a - x_b| - |x'_a - x'_b|. \quad (5.2)
$$

Before moving them, all points $p_i, \ldots, p_k$ are contained in the vertical stripe of $p_j$. The weight of $P'$ before the move is therefore at least $|x'_i - x| + |x'_k - x_k|$, while afterwards the $x$-coordinates of $p_i, \ldots, p_k$ are all equal to $x' \in \{x_l, x_r\}$. Thus, eq. (5.2) is at least

$$
|x'_i - x| + |x'_k - x_k| - |x'_i - x'| - |x'_k - x'|. \quad (5.3)
$$

If $p_i'$ and $p_k'$ lie on the same side of the vertical stripe of $p_j'$, the new coordinate $x'$ is closer to both $x_i$ and $x_k$. If $p_i'$ and $p_k'$ lie on different sides of the vertical stripe, then $|x'_i - x'| + |x'_k - x'| = |x'_i - x_k|$. In both cases eq. (5.3) is positive, and thus the weight of $P$ does not increase and the new placement is still optimum.

The case where $R_i = R_s$ is the same as the previous one if we define $p_i'$ to be the point $p_i$. The distance between $p_i$ and $p_i'$ is then always equal to 0, and the direction in which the points in a stripe are moved depends only on the position of $p_k'$. The same holds also for the remaining cases.

We now turn to the task of computing a minimum placement by exploiting the structural properties established in Lemma 5.2. Such a placement is constructed from a shortest path between two designated vertices in an auxiliary graph $D = (V_D, E_D)$ defined as follows. The set $V_D$ contains a vertex for every Hanan grid point that lies on the boundary of a polygon in $\mathcal{R}$, and two additional vertices $v_s$ and $v_t$ (see Figure 5.4 for an example). We introduce in $E_D$ an edge from $v_s$ to every grid point on the boundary of $R_s$ and from every grid point on the boundary of $R_t$ to $v_t$. Furthermore, let $u$ be a vertex of $R_i \in \mathcal{R}$ and $R_j$ be a polygon such that $(R_i, R_j) \in E$. For every segment on the boundary of $R_j$ we introduce in $E_D$ an edge from $u$ to its closest vertex on that
Min Shortest Path

Figure 5.4: The vertices of $D$ and the path $Q_D$ of Theorem 5.3 (in blue).

segment. We also introduce an edge from $u$ to the next vertex along the boundary of $R_i$ in both directions. We set the weight of $(u, v) \in E_D$ to 0 if $u = v_s$ or $v = v_t$ and to $\|u - v\|$ otherwise. The following theorem shows how to construct a minimum placement from a shortest $v_s$-$v_t$ path in $D$.

**Theorem 5.3.** Let $P_D$ be a shortest $v_s$-$v_t$ path in $D$ and $p$ be a placement as follows. If $R_i \in R$ contains a vertex along $P_D$, then $p_i \in p$ is the first of them; otherwise, $p_i$ is a corner of $R_i$ chosen arbitrarily. Then, $p$ is minimum.

**Proof.** Consider the vertices on $P_D$ that are in $p$ in the order as they appear on $P_D$. Since the polygons of these points are connected in $G$, $P_D$ corresponds to an $s$-$t$ path $P$ in $G$. By triangle inequality, the weight of $P$ in $G_p$ is at most the weight of $P_D$ in $D$. For the sake of contradiction, suppose there exists a placement $q$ for which a shortest $s$-$t$ path $Q$ in $G_q$ has weight smaller than that of $P$ in $G_p$. Without loss of generality we assume the points in $q$ to satisfy Lemma 5.2. Thus, every point of $q$ corresponds to a vertex of $D$. We construct a $v_s$-$v_t$ path $Q_D$ in $D$ as in Figure 5.4 as follows. The first and last edges are respectively $(v_s, q_s)$ and $(q_t, v_t)$. In between, for every edge $(R_i, R_j)$ on $Q$ we consider the points $q_i, q_j \in q$, the bounding box $B_{q_i q_j}$, and the at most two segments on the boundary of $R_j$ on which $q_j$ lies. By construction $q_i$ is connected in $D$ to a vertex on both segments; let $v$ be one of them chosen arbitrarily such that $v \in B_{q_i q_j}$. We insert in $Q_D$ the path from $q_i$ to $q_j$ passing through $v$ and following the boundary of $R_j$. By Fact 5.1, the weight of this path is $\|q_i - q_j\|$. To see that the weight of $Q_D$ is equal to that of $Q$ in $G_q$ note that the first and the last edge of $Q_D$ have weight 0 and, for every edge $(R_i, R_j)$ on $Q$, the sub-path of $Q_D$ from $q_i$ to $q_j$ has weight equal to $\|q_i - q_j\|$. This is a contradiction,
because $Q_D$ is then a path shorter than $P_D$. \qed

The above theorem shows how to construct a minimum placement once a shortest $v_s$-$v_t$ path in $D$ is known. We can compute such a path using Dijkstra’s algorithm in time $O(|V_D| \log |V_D| + |E_D|)$. To evaluate the quantities $|V_D|$ and $|E_D|$, consider a line of the Hanan grid. Each time this line intersects an orthogonal segment on the boundary of a polygon, an additional vertex is introduced. Conversely, each segment on the boundary of a polygon can in the worst case be intersected by every grid line orthogonal to it. If $k$ is the maximum number of corners of a polygon in $\mathcal{R}$ (and therefore on the number of segments of its boundary), the number of grid lines is $O(nk)$. Thus, the number of grid points lying on the boundary of a polygon is $O(nk^2)$ and the size of $V_D$ is $O(n^2k^2)$. To evaluate $|E_D|$, consider an edge $(R_i, R_j) \in E$ and a vertex $v$ of $R_i$. By construction, there is an edge from $v$ to a vertex on each of the at most $k$ segments on the boundary of $R_j$ and at most two edges connecting $v$ to vertices on $R_i$. Since each polygon contains at most $O(nk^2)$ vertices, the size of $E_D$ is $O(mnk^3)$. Thus, Dijkstra’s algorithm computes a shortest $v_s$-$v_t$ path in $D$ in time $O(n^2k^2 \log nk + mnk^3)$.

5.2 Max Shortest Path

We show hardness of approximation of MAX-SPP with a gap-producing reduction from 3-SAT. Given a 3-SAT instance, we construct a MAX-SPP instance and provide two threshold values $\tau_1, \tau_2$ with $\tau_1 - \tau_2 = 2\delta$ for an arbitrary constant $\delta > 0$. We show that the MAX-SPP instance admits a solution with weight at least $\tau_1$ if and only if the 3-SAT formula admits a satisfying assignment. If the formula does not admit a satisfying assignment, the weight of any solution is at most $\tau_2$.

In the reduction we make use of degenerate polygons, i.e., points. Note that in any placement the weight of an edge connecting two degenerate polygons is fixed. We will make use of this observation in the construction of the MAX-SPP instance when assuming that the weight of an edge connecting two degenerate polygons $p, q$ can be set to any weight greater or equal than the distance between $p$ and $q$. This can easily be achieved by splitting the edge and introducing an intermediate point at a suitable distance from both.

The $m$ clauses and $n$ variables of the 3-SAT formula are represented in
the MAX-SPP instance with the gadgets sketched in Figure 5.5. There is a gadget for every clause, one for every variable, and two additional points $s, t$. The point $s$ is located at the origin of the plane and $t$ is at coordinates $(3, 0)$.

The MAX-SPP instance contains $m + n$ vertically aligned segments. The middle points of these segments are equally spread along the $x$-axis, with the leftmost and rightmost ones respectively at coordinates $(1, 0)$ and $(2, 0)$. Given an arbitrary constant $\delta > 0$, the first $m$ segments have length $4\delta$ and each of them corresponds to a clause of the 3-SAT formula. The remaining $n$ segments have length $2\delta$ and they correspond to the variables. We use $v_i$ to denote both the $i$-th variable and the segment associated with it; it will be clear from the context to which of the two we are referring. Similarly, we use $c_j$ to denote both the $j$-th clause and its corresponding segment.

**Variable gadgets.** Given a segment $v_i$ at $x$-coordinate $x_i$, we place a point to its right, at coordinates $(x_i + \frac{1}{4\gamma}, 0)$, and two points to its left,
at coordinates \((x_i - \frac{1}{4\gamma}, -\delta), (x_i - \frac{1}{4\gamma}, \delta)\), where \(\gamma := m + n\). We use \(t_i\) and \(f_i\) to denote respectively these last two points. We introduce in the underlying graph edges connecting \(v_i\) to each of these three points. The point to the right of \(v_i\) is connected to \(t\) with an edge of weight 2. The points \(t_i\) and \(f_i\) are connected in a way specified in the following to gadgets of clauses where \(v_i\) appears as a literal. In the reduction, a point placed in the bottom half of \(v_i\) corresponds to assigning the value of “true” to \(v_i\). Conversely, a point in the top half of \(v_i\) corresponds to assigning the value of “false” to \(v_i\). Without loss of generality, we assume the middle point of any segment to be part of its top half.

**Clause gadgets.** Given a segment \(c_j\) at \(x\)-coordinate \(x_j\), we place 5 points next to it, at coordinates \((x_j - \frac{1}{4\gamma}, -\delta), (x_j - \frac{1}{4\gamma}, \delta), (x_j + \frac{1}{4\gamma}, -2\delta), (x_j + \frac{1}{4\gamma}, 0), (x_j + \frac{1}{4\gamma}, 2\delta)\) and introduce in the underlying graph edges connecting \(c_j\) to each of them. Among these points, those located to the left of \(c_j\) are connected to \(s\) with edges with weight \(2 + \delta\), while the others are connected to variable gadgets as follows. We uniquely associate the points at coordinates \((x_j + \frac{1}{4\gamma}, -2\delta), (x_j + \frac{1}{4\gamma}, 0), (x_j + \frac{1}{4\gamma}, 2\delta)\) with the variables appearing in \(c_j\) as literals. If \(v_i\) appears in \(c_j\) as a positive literal, we connect the point associated to it to the point \(f_i\) of the variable gadget of \(v_i\). If \(v_i\) appears in \(c_j\) as a negative literal, we connect the point associated with it to \(t_i\). We set the weight of these edges to \(2 + 3\delta\).

**Theorem 5.4.** For any metric \(L_p\), \(p \geq 1\), it is NP-hard to approximate the MAX-SPP for any factor \((1 - \epsilon)\) with \(\epsilon < \frac{1}{4}\).

**Proof.** We look at the case where distances are measured with the \(L_1\) metric first. Given an instance of 3-SAT, we create a MAX-SPP instance containing a gadget for every clause and every variable of the formula. Then, we consider every path between the points \(s\) and \(t\) in the underlying graph of the constructed MAX-SPP instance and provide bounds on its weight in any placement.

Observe that any \(s\)-\(t\) path passes through a segment \(v_i\) and one of the clauses where \(v_i\) appears as a literal. By construction of the variable gadgets, we can assume without loss of generality the point in \(v_i\) in any placement to be located either in one of its endpoints or in its middle point.

Suppose there exists a satisfying assignment for the 3-SAT instance. We
then place a point in the top endpoint of a segment $v_i$ if its value in the assignment is false and in the bottom otherwise. If a clause $c_j$ is satisfied by a variable $v_i$, we place in $c_j$ a point at the same $y$-coordinate as the point to the right of $c_j$ associated with $v_i$. If $c_j$ is satisfied by more than one variable, we select one of them arbitrarily.

Consider now any $s$-$t$ path through a clause $c_j$ and a variable $v_i$. If $v_i$ is the chosen variable for $c_j$, the weight of the path between a point to the left of $c_j$ to the point to the right of $c_j$ associated with $v_i$ is at least $\delta + \frac{1}{2\gamma}$. Without loss of generality let $v_i$ appear with positive sign in $c_j$. Since the path visits $f_i$ and the point in $v_i$ is located at its bottom, its weight between $f_i$ and the point to the right of $v_i$ is at least $3\delta + \frac{1}{2\gamma}$. Since the remaining edges of the path have fixed weight, the overall weight of the $s$-$t$ path in the placement is

$$
\tau_1 := 6 + 8\delta + \frac{1}{\gamma}.
$$

If $v_i$ is not the chosen variable for $c_j$, the weight of the path between the point to the left of $v_i$ and the one to its right might be smaller, but still at least $\delta + \frac{1}{2\gamma}$. On the other hand, the weight of the path between the point to the left of $c_j$ and the one to its right is at least $3\delta + \frac{1}{2\gamma}$. Also in this case, the weight of the $s$-$t$ path is therefore at least $\tau_1$.

Suppose now that a satisfying assignment does not exist and consider an optimum placement and the corresponding assignment of variables according to whether the point in $v_i$ lies in its top or bottom half. Since the formula is not satisfiable, we can always find in this assignment a non satisfied clause $c_j$. It is easy to see that there exists an $s$-$t$ path through $c_j$ and a variable appearing in $c_j$ with weight at most

$$
\tau_2 := 6 + 6\delta + \frac{1}{\gamma}.
$$

To conclude, assume that there exists a polynomial time algorithm approximating MAX-SPP for some factor $(1 - \epsilon)$ with $\epsilon < 1/4$. For any instance of 3-SAT, we can then construct the above gadgets and calculate $\tau_1$ and $\tau_2$. Since $\tau_1 - \tau_2 = 2\delta$, setting $\delta$ such that $\frac{2\delta}{\tau_1} > \epsilon$ and using the approximation algorithm, we would be able to decide in polynomial time whether the weight of an optimum solution is greater or smaller respectively than $\tau_1$ or $\tau_2$. Thus, we could determine in polynomial time whether there exists a satisfying assignment for the 3-SAT instance.
In the case where distances are measured with a general metric $L_p, p > 1$ similar bounds hold. In particular, it can be shown that, if the 3-SAT formula is satisfiable, the weight of any $s$-$t$ path in an optimum placement is at least

$$
\tau_1 := 6 + 6\delta + \frac{1}{2\gamma} + 2^{p/4} \left( \frac{1}{4\gamma} \right)^p + \delta^p.
$$

If the formula is not satisfiable, the weight of a shortest path in an optimum placement is at most

$$
\tau_2 := 6 + 4\delta + \frac{1}{2\gamma} + 2^{p/4} \left( \frac{1}{4\gamma} \right)^p + \delta^p.
$$

We observe that the result of Theorem 5.4 comes from the fact that the ratio $\frac{2\delta}{\tau_1}$ is roughly $\frac{1}{4} \left( \frac{\delta}{\delta+1} \right)$. It is an interesting open problem to determine whether there exist an algorithm with approximation factor exactly $3/4$ (therefore making the above bound tight), or whether the problem cannot be approximated even for values of $\epsilon$ larger or equal than $1/4$.

### 5.3 Max Touring Polygons

In the maximization version of the Touring Polygons Problem (or MAX-TPP for short), we are given an ordered sequence of polygonal regions $P_1, \ldots, P_n$ and two points $s, t$. We are asked for a longest tour that starts at $s$, visits the polygons in the given order, and ends at $t$. The MAX-TPP is a special case of MAX-SPP where the underlying graph connecting the polygons is a path. We show that, contrary to the general case, the MAX-TPP can be solved in polynomial time even if the polygons are non-convex and overlapping. Our algorithm is based on the observation that the points belonging to an optimum placement form a subset of the corners of the $n$ polygons. This observation is proved by the following lemma. In the following, we assume that distances are measured with a metric $L_p$ with $p \geq 1$.

**Lemma 5.5.** There exists an optimum placement for the MAX-TPP where every point is a corner of the corresponding polygon.
Proof. We prove the statement by showing that, given a placement resulting in a longest $s$-$t$ tour (an optimum placement) we can construct an equivalent placement in which every point is a corner of its respective polygon.

First, we show that there always exists an optimum placement where no point lies in the interior of the corresponding polygon. For the sake of contradiction, assume there is a point $p_i \in p$ lying in the interior of $P_i$ and the points $p_{i-1}$ and $p_{i+1}$ (where $p_0 := s$ and $p_{n+1} := t$). If $p_{i-1} = p_i$, moving $p_i$ away from its position in any direction would not decrease the weight of the optimum placement. If $p_{i-1} \neq p_i$, consider the ray from $p_{i-1}$ towards $p_i$. This ray crosses a point $q$ on the boundary of $P_i$ after crossing $p_i$. Furthermore, the triangle induced by $p_{i-1}, p_i, p_{i+1}$ is completely contained in the triangle induced by $p_{i-1}, q, p_{i+1}$ (see Figure 5.6). Therefore, by triangle inequality the weight of the tour does not decrease if we replace $p_i$ with $q$. We can repeat this operation for every point of $p$ until every $p_i \in p$ lies on the boundary of the corresponding polygon $P_i$.

Given an optimum placement $p$ where every point lies on the boundary of the corresponding polygon, we show how to construct an optimum placement where every point lies on a corner. Consider a point $p_i \in p$ not lying on a corner of $P_i$ and the segment $\overline{ab}$ of the boundary of $P_i$ on which $p_i$ lies. Since the sum $\|p_{i-1} - r\|_p + \|r - p_{i+1}\|_p$ is convex for $r \in \overline{ab}$, its maximum is attained when $r$ is one of the endpoints; assume without loss of generality that it is maximized when $r$ is equal to $a$. Therefore, the weight of the tour does not decrease if we replace $p_i$ with $a$ (that is a corner of $P_i$). By repeatedly applying this statement, we can replace every point of $p$ not lying on a corner until every $p_i \in p$ lies on
Using Lemma 5.5 we can design an algorithm to compute an optimum placement for the MAX-TPP. For this purpose we construct an auxiliary graph \( D = (V_D, E_D) \), in which there is a vertex \( v \in V_D \) for every corner of a polygon \( P_i \) and two special vertices \( v_s, v_t \in V_D \) corresponding respectively to \( s \) and \( t \) (see Figure 5.7). There is an edge \( (u, v) \in E_D \) for every vertex \( u \) corresponding to a corner of \( P_i \) and every vertex \( v \) corresponding to a corner of \( P_{i+1} \), for \( i = 1, \ldots, n \). Furthermore, there is an edge from \( v_s \) to corner of \( P_1 \) and to \( v_t \) from every corner of \( P_n \). We assign a weight to each edge equal to the distance between its endpoints. The following theorem shows that a longest path between \( v_s \) and \( v_t \) in \( D \) corresponds to an optimum MAX-TPP placement.

**Theorem 5.6.** Let \( P = \langle u_0, \ldots, u_{n+1} \rangle \) with \( u_0 = v_s \) and \( u_{n+1} = v_t \) be a longest \( v_s-v_t \) path in \( D \) and \( p \) be the placement where \( p_i \in p \) corresponds to \( u_i \) for every \( i = 1, \ldots, n \). Then, \( p \) is maximum.

**Proof.** Observe that the weight of \( P \) is equal to the weight of the tour induced by \( p \). For the sake of contradiction, assume there exists a placement \( q \) inducing a tour that is longer than the one induced by \( p \). By Lemma 5.5 we can assume the points of \( q \) to be corners of the polygons \( P_1, \ldots, P_n \). Thus, the vertices corresponding to the points of \( q \) are connected in \( D \), and we can find a path \( Q \) from \( v_s \) to \( v_t \) following these vertices of weight larger than that of \( P \). \( \Box \)

Computing longest paths is in general NP-hard. Observe however that \( D \) contains edges only from vertices of a polygon to vertices of the successive one. Therefore, \( D \) is a DAG (directed acyclic graph), and we can compute a longest \( v_s-v_t \) path with a trivial dynamic programming
algorithm in time $O(|V_D| + |E_D|)$. To evaluate $|V_D|$ and $|E_D|$, let $k$ be the largest number of corners of a polygon $P_1, \ldots, P_n$. Since $D$ contains a vertex for every corner, one for $s$, and one for $t$, we get $|V_D| \leq n \cdot k + 2 \in O(n \cdot k)$. Furthermore, there is an edge from each corner of $P_i$ to every corner of $P_{i+1}$, for $i = 0, \ldots, n$, with $P_0$ and $P_{n+1}$ being degenerate polygons corresponding to $s$ and $t$. Therefore, every vertex of $V_D$ is adjacent to at most $k$ edges, that is, $|E_D| \in O(n \cdot k^2)$. The time to find a maximum placement for the MAX-TPP is thus $O(n \cdot k^2)$. 
We define the weight of a tree as the sum of the lengths of its edges and its diameter as the length of its longest simple path. A classical problem in computational geometry is the computation of a tree connecting all points having minimum weight or diameter. We consider the problem of computing such minimum spanning trees in the imprecise points model. We are given a set of regions in the plane and our task is to place a point inside each region such that the minimum spanning tree in terms of either weight or diameter of the resulting point set is as small as possible (see Figure 6.1 for examples).

Löffler and van Kreveld [94] prove the problem of computing the smallest weight of a spanning tree for imprecise points in non-necessarily disjoint disks or squares to be NP-hard. Dorrigiv et al. [42] strengthen the result by showing that the minimization as well as the maximization variant of the problem do not admit a fully-polynomial time approximation scheme (FPTAS) if the regions consist of disjoint disks. The authors note that it would be interesting to study the problem with different shapes of imprecise points, like line segments. In Section 6.1 we prove the minimization variant to not admit a FPTAS even if the segments are only horizontally or vertically aligned. Concerning the maximization variant, we observe that Dorrigiv et al. consider in their proof only two possible locations for the placement of a point in each disk. Replacing the disk with a segment connecting these two points generalizes the hardness of the maximization variant to the case of axis-aligned line segments as well. Finally, Yang et al. [130] show a PTAS for
6.1 Minimum Weight Spanning Trees

We show that the minimization variant of the minimum weight spanning tree problem for imprecise points (in short MWST) does not admit an FPTAS with a reduction from planar 3-SAT. Planar 3-SAT is a variant of 3-SAT where the graph associated with the formula is planar. The graph contains a vertex for each variable and each clause; there is an edge connecting a variable to a clause if the clause contains a literal of that variable. Planar 3-SAT was shown to be \(\text{NP}\)-hard by Lichtenstein [87]. Knuth and Raghunathan [81] observed that the graph of a planar 3-SAT formula admits a rectilinear plane embedding where the variables are arranged on a straight line, called the spinal path, and the clauses are drawn as three legged segments completely above or below them. Figure 6.2 shows an example of such an embedding.
Minimum Weight Spanning Trees

Figure 6.2: A planar 3-SAT instance on 5 variables. Dashed lines are parts of the spinal path, solid lines are clauses.

Given a planar 3-SAT formula with $n$ variables and $m$ clauses, we construct an instance of the MWST and provide two thresholds $\tau_1, \tau_2$ with $\tau_2 - \tau_1 = \delta$ for an arbitrary constant $\delta$. We show that the MWST instance admits a solution with weight at most $\tau_1$ if and only if the formula is satisfiable. If the formula does not admit a satisfying assignment, we show that the weight of any solution is at least $\tau_2$. In the following we will assume that distances are measured using a metric $L_p$ with $p \geq 1$.

The MWST instance follows the embedding of the planar 3-SAT formula, replacing variables, clauses and the spinal path each with its own gadget. An important tool in the construction of the gadgets is a so-called wire. A wire is a sequence of degenerate polygons (i.e., points) placed in close succession one to the other. Since any minimum weight spanning tree of a point set contains all the edges with minimum weight, if they do not form a cycle, if the points in a wire are at a suitably small distance to each other we are guaranteed that the wire will be a path in the minimum spanning tree of any placement. However, this suitably small distance must still be large enough to ensure the size of a wire to be polynomial. Since in the construction the smallest non-zero distance between any two polygons other than those in the wires will be $\delta/2$, a suitably small distance between two points in a wire is, for example, $\delta/4$.

Variable gadget. For each variable we introduce $6m + 6$ segments of length $\alpha$ (we specify the concrete value of $\alpha$ in the following), placed along the perimeter of a rectangle with sides of length $3m\alpha + \delta$ and $3\alpha + \delta$. In the interior of this rectangle we place a set of wires connected together in a tree-like structure. There is a wire for every two consecutive segments in clockwise order. Each of these wires terminates on
the line bisecting the angle formed by the corresponding segments. For parallel segments, the endpoint is at distance $\delta$ from their common point. For perpendicular segments, the endpoint is at distance $\delta$ from the intersection of the lines passing through them. Figure 6.3 shows an example of a variable gadget. The variable gadgets are at distance $5\delta$ from each other and such that their vertical middle point lies on the $x$-axis.

We call a placement of points for a variable gadget a configuration if it contains either the two closest points or the two farthest points for any two consecutive segments in clockwise order. For a variable gadget there exist exactly two different configurations; we associate one of them with the assignment of “true” to the variable and the other with the assignment of “false”.

**Clause gadget.** For every clause we introduce at most three wires meeting at a single point following the embedding. As in Figure 6.3, each wire of a clause gadget approaches the common point of two horizontal segments of a variable appearing in the clause as a literal. Clauses located above the spinal path in the rectilinear embedding approach variable gadgets from above, while clauses located in the embedding below the spinal path approach variable gadgets from below. Furthermore, clause wires approach a variable gadget in the same clockwise order as the edges connecting the variable vertex to the corresponding clauses in the rectilinear embedding. We place these wires such that the distance between any two points in wires of different clauses is at least $3\alpha$. Each wire of a clause gadget terminates at distance $3\delta$ from the common point of the approached segments along the vertical line passing through it. The approached segments are chosen such that their common point is part of a configuration of the variable gadget satisfying the clause. That is, an edge with cost $3\delta$ connects the clause wire to the segments if the variable is in a configuration satisfying the clause.

**Spinal path gadget.** This gadget consists of wires following the $x$-axis according to the embedding of the planar 3-SAT instance. As in Figure 6.3, these wires bifurcate when close to a variable gadget. Each part approaches two adjacent vertical segments of the variable gadget. The point at which a part terminates is at distance $2\delta$ from the common point of the approached segments along the horizontal line passing
Theorem 6.1. For any metric $L_p$, $p \geq 1$, the MWST does not admit an FPTAS.

Proof. We first create an instance of MWST using the above gadget from a rectilinear embedding of an instance of planar 3-SAT. Then, we provide bounds on the weight of a minimum spanning tree in a minimum placement according to whether the formula is satisfiable or not. We use $w_{\text{wires}}$ to denote the fixed weight of all the wires used in the construction.

Suppose there exists a satisfying assignment. We can then place points in each variable gadget in a configuration according to its value in the assignment. We provide an upper bound $\tau_1$ on the weight of an optimum placement for this case by constructing a spanning tree and evaluating its cost. For each pair of consecutive segments having their closest points in the placement, the spanning tree connects them to the wire in the interior of their variable gadget with an edge with weight $\delta$, resulting in a weight of $(3m + 3)\delta$ per variable. For each clause gadget we select a variable satisfying it in the assignment. We connect the

Figure 6.3: The gadget of a variable appearing in $c_k$ as a negative and in $c_j, c_l$ as a positive literal. Dashed lines are wires, solid lines are segments.
corresponding endpoint of the clause wire to one of the segments it approaches with an edge with weight $3\delta$. For each part of the spinal path approaching a variable, exactly one of its endpoint approaches a point of the placement. The spanning tree contains the $2n$ edges of weight $2\delta$ connecting them. The overall weight of the constructed tree is

$$\tau_1 := w_{\text{wires}} + (3m + 3)\delta n + 3\delta m + 4\delta n.$$ 

If there is no satisfying assignment we consider an optimum placement where every point in a segment is located in one of its extreme points. The existence of such a placement is guaranteed by the fact that any wire approaching a variable gadget terminates either to the left or to the right of horizontal segments, and above or below vertical segments. We first provide an upper bound on the weight of a minimum spanning tree for such a placement by constructing a spanning tree and evaluating its weight. We then use this bound to show that, in every minimum spanning tree, clause wires are connected to the tree either with an edge from one of the endpoints to one of the approached segments, or with an edge from one of its endpoints to an endpoint of the wire inside the approached variable. Finally, we use the proved structure of the spanning tree to show that the weight of any optimum solution is at least

$$\tau_2 := \tau_1 + \delta.$$ 

The spanning tree of which we evaluate the weight contains all the wires used in the MWST instance. Every segment in a variable gadget is connected to an endpoint of the wire inside that gadget with an edge of weight $\delta$. Every fork of the spinal path is connected to the approached variable it approaches by one of its endpoints with an edge with weight $2\delta$. An endpoint of each clause wire chosen arbitrarily is connected to the wire in the interior of the approached variable gadget with an edge with weight $4\delta$. The overall weight of such a tree is

$$w_{\text{wires}} + (6m + 6)\delta n + 4\delta m + 4\delta n. \quad (6.1)$$ 

We now prove that, in any minimum spanning tree, each clause wire is connected with an edge from one of its endpoints to either an approached segment or to the wire in the interior of the approached variable. Towards contradiction, suppose there is a clause not connected to the approached segment or to the wire in the approached variable.
The next closest object is by construction at distance at least \( \alpha \), the length of the segments used in the variable gadgets. Setting

\[
\alpha := (6m + 6)\delta n + 4\delta m + 4\delta n + 1
\]

we obtain a contradiction, because the weight of a minimum spanning tree would then be greater than eq. (6.1).

Finally, we show that the weight of an optimum solution is at least \( \tau_2 \) if the formula is not satisfiable. By this assumption, in any optimum solution either at least one variable is not in a configuration, or every variable is in a configuration and for at least one clause none of its wires approaches a point in the placement. In the former case, let \( j > 0 \) be the number of clauses that are not satisfied by the assignment corresponding to the configuration. The weight of a minimum spanning tree is at least

\[
w_{\text{wires}} + (3m + 3)\delta n + 3\delta(m - j) + 4\delta j + 4\delta n \geq \tau_2.
\]

In the latter case, there exists at least one variable gadget that is not in a configuration. We thus need an additional weight of at least \( \delta \) to connect all the segments of the variable gadgets to a minimum spanning tree, for a resulting overall weight of at least \( \tau_1 + \delta = \tau_2 \).

To conclude, assume there exists an FPTAS for the MWST. For any instance of planar 3-SAT, we can then construct the above gadgets and calculate \( \tau_1 \) and \( \tau_2 \). Setting the constant \( \delta \) such that \( \frac{\delta}{\tau_1} > \epsilon \) and using the FPTAS, we would be able to decide whether the weight of an optimum solution is smaller or greater respectively than \( \tau_1 \) or \( \tau_2 \). Thus, we could determine in polynomial time whether there exists a satisfying assignment for the planar 3-SAT instance.

Remark. The proofs of Theorem 5.4 and of Theorem 6.1 are conceptually similar. However, the former proves hardness of approximation within some constant factor, while the latter only proves that there does not exist an FPTAS for the problem. The reason for this difference is that for the former problem the ratio between the parameter \( \delta \) and the threshold \( \tau_1 \) does not change for increasing or decreasing values of \( n \) or \( m \). The same cannot be said for the latter problem, though. The ratio between \( \delta \) and \( \tau_1 \) becomes in this case infinitely small if \( n \) or \( m \) go to infinity.

Remark. The gadgets used above are almost identical to those of Dorrigiv et al. [42]. Our proof is however antecedent [40] to that particular publication. In the conference version of their result [41] Dorrigiv et
Consider a non-rectilinear embedding of the planar 3-SAT formula and therefore propose slightly different gadgets.

### 6.2 Minimum Diameter Spanning Trees

Let \( D = \{D_1, \ldots, D_n\} \) be a set of disks in the plane with centers \( c_i = (h_i, k_i) \) and radius \( r_i \) for \( i = 1, \ldots, n \). We study the task of placing a point inside each disk such that the diameter of a minimum diameter spanning tree of the resulting point set is minimized (to avoid trivialities we assume the intersection of all disks to be empty). We show that such a placement can be computed in polynomial time for general disks and present further improvements for various special cases. The following theorem summarizes our results.

**Theorem 6.2.** The minimum diameter of a spanning tree of \( n \) points each of which is selected from a respective disk region can be computed in \( O(n^9) \) time for arbitrary disks and in \( O(n^6) \) time for unit disks. If the disks are disjoint, the required computation times decrease respectively to \( O(n^8 \log^2 n) \) and \( O(n^5) \).

For points whose locations are all exactly known, Ho et al. [70] prove that there always exists a minimum diameter spanning tree that is either *monopolar*, i.e., it contains a point, called a *pole*, linked to all the remaining points, or *bipolar*, i.e., it contains two poles such that all remaining points are linked to one of the two, and developed an algorithm for computing such a tree in \( O(n^3) \) time. Chan [30] further improves the time bound to \( o(n^{17/6+\alpha}) \) for any fixed \( \alpha > 0 \). Since the best known exact algorithm still takes near-cubic time, some authors [62, 125] propose fast approximation algorithms.

Theorem 6.2 reveals a stark contrast: while for exact points a minimum weight spanning tree can be computed much faster than a minimum diameter spanning tree \( (O(n \log n) \) run-time versus almost cubic), the reverse is true for imprecise points (NP-hardness versus polynomial run-time). The difference in the latter setting comes from the fact that there always exists a minimum diameter spanning tree whose longest path consists of at most three edges. Therefore, the diameter can be attained by at most four disks, while the cost depends instead on all the \( n \) disks.

There are two main challenges for achieving polynomial run-time. First, although there are only linearly many monopolar tree topologies, the
Minimum Diameter Spanning Trees

number of bipolar topologies is exponential in the number of points. At a first glance, it is not clear how to reduce the number of topologies to consider without the exact locations of points. Second, even for a fixed tree topology, it is unknown how to place points in the disks efficiently in order to minimize the diameter. For the first challenge we use the arrangements formed by bisectors among disks to prove that it is sufficient to consider only $O(n^7)$ bipolar tree topologies. For unit disks we reduce this quantity to $O(n^4)$ topologies admitting a straight line separating the centers of the disks linked to one pole from the centers of the disks linked to the other pole. For the second challenge we employ first- and second-order farthest-disk Voronoi diagrams respectively for the bipolar and for the monopolar case to analyze the locations of poles and compute optimum locations for them. For disjoint disks we develop a sequential search achieving a faster run-time than in the intersecting case. We further improve the run-time for unit disks by computing farthest-disk Voronoi diagrams in a batch.

In our computation we will make use of two basic operations whose run-time is indicated by the following facts. These operations are implemented by the Newton-Raphson method and are described in detail in Section 6.3. For a point $x$ and a disk $D$, we let $d(x, D)$ denote the smallest distance between $x$ and $D$; in other words, $d(x, D) := \min_{y \in D} |xy|$. Furthermore, given a compact, connected subset $A \subset \mathbb{R}^2$, we use $\text{cl} A$ and $\partial A$ to indicate respectively the closure and the boundary of $A$.

**Fact 6.3.** For two disks $D_p, D_q$ and an edge $e$ that is a line segment or a circular or hyperbolic arc, a point $t \in \text{cl} e$ minimizing $d(t, D_p) + d(t, D_q)$ can be computed in $O(1)$ time.

**Fact 6.4.** For two disks $D_p, D_q$ and two circular arcs $e_1, e_2$, the points $s \in \text{cl} e_1, t \in \text{cl} e_2$ minimizing $d(s, D_p) + |st| + d(t, D_q)$ can be computed in $O(1)$ time.

**Farthest-disk Voronoi diagrams**

The *farthest-disk Voronoi diagram* $FV(D)$ of a set of disjoint disks $D$ is a planar subdivision such that all points in a region share the same farthest disk among $D$ (Figure 6.4a shows an example). We use $FVR(D, D)$ to denote the Voronoi region associated with $D \in D$. The common boundary between two Voronoi regions is called a Voronoi edge, and the common point among more than two Voronoi regions is called a Voronoi
Figure 6.4: (a) Farthest, (b) second-order farthest, and (c) refined second-order farthest Voronoi diagram of a set of points.

vertex. An equivalent definition of \( \text{FV}(\mathcal{D}) \) is using bisectors. For two disks \( D \) and \( D' \), the bisector \( B(D, D') = \{ x \in \mathbb{R}^2 \mid d(x, D) = d(x, D') \} \) partitions the plane into two connected regions \( H(D, D') = \{ x \in \mathbb{R}^2 \mid d(x, D) < d(x, D') \} \) and \( H(D', D) = \{ x \in \mathbb{R}^2 \mid d(x, D') < d(x, D) \} \). We then have

\[
\text{FVR}(D, \mathcal{D}) = \bigcap_{D' \in \mathcal{D} \setminus \{D\}} H(D', D), \quad \text{FV}(\mathcal{D}) = \bigcup_{D \in \mathcal{D}} \partial \text{FVR}(D, \mathcal{D}).
\]

If two disks \( D \) and \( D' \) intersect, their bisector contains a bi-dimensional face corresponding to \( D \cap D' \), which makes the definition of \( \text{FV}(\mathcal{D}) \) ambiguous. To eliminate the ambiguity, we amend the bisector system as follows. If \( D' \subseteq D \), we let \( H(D, D') = \mathbb{R}^2 \), and both \( B(D, D') \) and \( H(D', D) \) be empty. The treatment is symmetric if \( D \subseteq D' \). Otherwise,
Figure 6.5: The amended bisectors.

let \( y, z \) be the intersection points between \( \partial D \) and \( \partial D' \); we redefine \( B(D, D') \) as \( \{ x \in \mathbb{R}^2 \setminus D \cap D' \mid d(x, D) = d(x, D') \} \cup \overline{yz} \). That is, we replace the part of \( B(D, D') \) within \( D \cap D' \) with \( \overline{yz} \). The sets \( H(D, D') \) and \( H(D', D) \) become the connected regions induced by the resulting curve and containing their original parts. Figure 6.5 illustrates the amended bisectors.

The second-order farthest-disk Voronoi diagram \( FV_2(D) \) is a planar subdivision such that all points in a region share the same pair of farthest disks. For our purposes, we will require each region of \( FV_2(D) \) to be further partitioned according to which of the two disks associated with it is the farthest and which is the second-farthest. We call this refinement the refined second-order farthest-disks Voronoi diagram and denote it as \( FV_2^R(D) \). Figures 6.4b and 6.4c show examples of second-order and refined second-order farthest Voronoi diagrams. Defining \( FV_R(D, D) \) as the intersection between \( FV(D, D) \) and \( FV(D', D \setminus \{D\}) \), we get

\[
FV_2^R(D) = \bigcup_{D, D' \in \mathcal{D}} \partial FV_R((D, D'), D).
\]

All points in \( FV_R(D, D) \) share the same farthest disk \( D \) and the same second farthest disk \( D' \) in \( \mathcal{D} \). The following theorems provide structural and computational properties of \( FV(D) \) and \( FV_2(D) \).

**Theorem 6.5.** \( FV(D) \) and \( FV_2(D) \) have \( O(n) \) faces, and \( FV(D) \) is a tree.

**Proof.** Mehlhorn et al. [100] prove that if the bisecting curve system satisfies several properties, the resulting farthest-site Voronoi diagram is a tree and has \( O(n) \) faces. After removing from \( \mathcal{D} \) all disks that contains another disk, the bisectors among the remaining disks will satisfy these properties. Note that, using the amended bisectors, if a disk \( D \) contains
another one then \( \text{FV}(D, D) \) is empty; thus, the removal of \( D \) will not change \( \text{FV}(D) \), implying that \( \text{FV}(D) \) has \( O(n) \) faces and is a tree.

To analyze the structural properties of \( \mathcal{FV}_2(D) \) we introduce the \( k \text{-th} \) order nearest-site and farthest-site Voronoi diagrams. Given a set \( S \) of \( n \) sites, the \( k \text{-th} \) order nearest-site Voronoi diagram \( V_k(S) \) is a planar subdivision such that all points in a region share the same \( k \) nearest sites among \( S \), and the \( k \text{-th} \) order farthest-site Voronoi diagram \( \text{FV}_k(S) \) is a planar subdivision such that all points in a region share the same \( k \) farthest sites among \( S \). Since sharing the same \( k \) nearest sites is equivalent to sharing the same \( n-k \) farthest ones, it holds \( V_k(S) = \text{FV}_{n-k}(S) \). We let \( \text{FVR}_k(Q, S) \) denote the Voronoi region of \( \text{FV}_k(S) \) associated with a \( k \)-element subset \( Q \) of \( S \).

Bohler et al. [18] prove that if the bisecting curve system satisfies several properties, \( V_k(S) \) has at most \( 2k(n-k) \) faces, and so does \( \text{FV}_k(S) \). Although bisectors among disks do not satisfy all those properties (because a first-order nearest-site Voronoi region can be empty), certain properties [18, Lemmas 6, 7, 12, 13, 14] still hold. Those properties imply that the number of faces of \( \mathcal{FV}_2(D) \) is at most twice the number of faces of \( \text{FV}(D) \). To compare the size of \( \mathcal{FV}_2(D) \) and of \( \mathcal{FV}_2(D) \) note that, for any two disks \( D \) and \( D' \), the regions \( \text{FVR}_2((D, D'), D) \) and \( \text{FVR}_2((D', D), D) \) form a partition of \( \text{FVR}_2(\{D, D\}', D) \) by \( B(D, D') \). Thus, the number of faces in \( \mathcal{FV}_2(D) \) is linear in the number of faces of \( \mathcal{FV}_2(D) \).

**Theorem 6.6.** \( \text{FV}(D) \) and \( \mathcal{FV}_2(D) \) can be computed in \( O(n \log^2 n) \) and \( O(n^2 \log^2 n) \) time, respectively. If \( D \) consists of unit disks, both \( \text{FV}(D) \) and \( \mathcal{FV}_2(D) \) can be computed in \( O(n \log n) \) time.

**Proof.** Cheong et al. [31] proposes a divide-and-conquer algorithm computing farthest-polygon Voronoi diagrams in \( O(n \log^3 n) \) time. If we first remove from \( D \) all disks containing another one (for example, with a simple plane-sweep technique running in \( O(n \log n) \) time), the algorithm is also applicable to \( D \), even under the modified bisectors. Since a disk containing another one does not have regions in \( \text{FV}(D) \), we can thus compute \( \text{FV}(D) \) in \( O(n \log^3 n) \) time. Moreover, one of the \( O(\log n) \) factors in the run-time comes from the handling of closed merge curves. In the “divide” stage, if we separate the disks according to the \( x \)-coordinates of their centers, there will be no closed merge curve at all. Therefore, \( \text{FV}(D) \) can be computed in \( O(n \log^2 n) \) time. To compute \( \mathcal{FV}_2(D) \) we further partition each nonempty \( \text{FV}(D, D) \) ac-
Minimum Diameter Spanning Trees

103

According to $\text{FV}(\mathcal{D} \setminus \{D\})$. Since $\text{FV}(\mathcal{D})$ has $O(n)$ faces, the computation of $\mathcal{FV}_2(\mathcal{D})$ takes $O(n) \cdot O(n \log^2 n) = O(n^2 \log^2 n)$ time. Finally, if $\mathcal{D}$ consists of unit disks, then $\text{FV}(\mathcal{D}) = \text{FV}(\mathcal{C})$ and $\mathcal{FV}_2(\mathcal{D}) = \mathcal{FV}_2(\mathcal{C})$. Ho et al. [70] showed how to compute these diagrams in $O(n \log n)$ time.

Monopolar Case

We now consider the case where a minimum placement for $\mathcal{D}$ admits a minimum diameter spanning tree that is monopolar (in short we say that $\mathcal{D}$ admits monopolarity). For a point $p_i \in D_i$, we use $\delta_i$ to denote the smallest diameter that the monopolar spanning tree with pole $p_i$ can achieve by placing points in $\mathcal{D} \setminus \{D_i\}$. The general idea for this case is to compute an optimum pole $p_i$ for each disk $D_i \in \mathcal{D}$ and select the one resulting in the smallest diameter. Observe that, for a fixed pole $p_i$, for every disk $D_j \in \mathcal{D} \setminus \{D_i\}$ is sufficient to place $p_j$ at the first intersection between $\overrightarrow{p_i c_j}$ and $D_j$. The following fact relates $p_i$ to the refined second-order farthest Voronoi diagram $\mathcal{FV}_2(\mathcal{D})$. Using this fact, we prove that it is sufficient to consider as locations for $p_i$ only $\partial D_i$ and the edges of $\mathcal{FV}_2(\mathcal{D})$.

**Fact 6.7.** For a pole $p_i \in D_i$, it holds that

$$\delta_i = d(p_i, D_j) + d(p_i, D_k),$$

where $D_j$ and $D_k$ are respectively the first and second farthest disks of $p_i$ in $\mathcal{D}$.

**Lemma 6.8.** If $\mathcal{D}$ admits monopolarity, there exists a minimum placement where the pole $p_i$ belongs to $\partial D_i$ or $\mathcal{FV}_2(\mathcal{D}) \cap D_i$.

**Proof.** Using the notation of Fact 6.7, it holds $p_i \in \mathcal{FVR}_2((D_j, D_k), \mathcal{D})$. Let $p'_i$ be the first intersection between $\partial(\mathcal{FVR}_2((D_j, D_k), \mathcal{D}) \cap D_i)$ and $\overrightarrow{p_i c_j}$. Since $D_j$ is the farthest disk for all points in $\overrightarrow{p_i p'_i}$, the segment $p_i p'_i$ does not intersect $D_j$; otherwise, all disks contain the intersection point, contradicting the assumption that the intersection of all disks is empty. Since moving $p_i$ toward $c_j$ will not increase $d(p_i, D_j) + d(p_i, D_k)$ unless $p_i$ enters $D_j$ (no matter whether $p_i$ belongs to $D_k$ or not), replacing $p_i$ with $p'_i$ results in a placement satisfying the statement. To conclude, note that $\mathcal{FVR}_2((D_j, D_k), \mathcal{D}) \cap D_i = \emptyset$ implies $\partial(\mathcal{FVR}_2((D_j, D_k), \mathcal{D}) \cap D_i) = \partial D_i$. □
To compute an optimal pole in $D_i$ it is thus sufficient to consider the planar graph formed by combining $\partial D_i$ with the edges of $\mathcal{FV}_2(D) \cap D_i$. For each edge $e$ of the planar graph we find the optimum location of $p_i$ on $\text{cl} e$ and select the one resulting in the smallest diameter, concluding the following.

**Theorem 6.9.** If $D$ admits monopolarity, a minimum placement can be computed in $O(n^2 \log^2 n)$ time and in $O(n^2)$ time if all disks are unit.

**Proof.** Consider a disk $D_i$ and an edge $e$ of the planar graph induced by $\partial D_i$ and $(\mathcal{FV}_2(D) \cap D_i)$. If $e \subset \text{cl} \mathcal{FV}_2 \left( \left( D_j, D_k \right), D \right)$, by Fact 6.7, the optimum location $p_i \in \text{cl} e$ minimizes $d(p_i, D_j) + d(p_i, D_k)$. Furthermore, if $e$ belongs to $\partial D_i$, then $e$ is a circular arc; otherwise, $e$ belongs to a bisector between two disks and consists of at most three parts, each of which is a line segment or a hyperbolic arc. By Fact 6.3 we can then compute $p_i$ in $O(1)$ time. By Theorem 6.5 the planar graph has $O(n)$ edges, and thus an optimal location $p_i \in D_i$ can be computed in $O(n)$ time, resulting in an overall $O(n^2)$ time for all $n$ disks. Furthermore, by Theorem 6.6 we can compute $\mathcal{FV}_2(D)$ in $O(n^2 \log^2 n)$ and in $O(n \log n)$ time respectively for general and unit disks, leading to the statement.

**Bipolar Case**

We now describe how to compute a minimum placement in the case where $D$ does not admit monopolarity. It is known [70] that in this case there exists a minimum diameter spanning tree that is bipolar. For the remainder of this section we use $p_i \in D_i$ and $p_j \in D_j$ to denote the two poles and $D_i$ and $D_j$ to denote the disks of $D \setminus \{D_i, D_j\}$ whose points are linked respectively to $p_i$ and to $p_j$. We also call $D_i$ and $D_j$ a pair of polar disks and a configuration consisting of $D_i, D_j, D_i, D_j$ a bipolar tree topology. For a fixed topology, $\delta_{ij}$ denotes the smallest diameter that the corresponding tree can achieve by placing points in $D \setminus \{D_i, D_j\}$. Observe that, for fixed poles $p_i$ and $p_j$, it is sufficient to place $p_k$ and $p_l$ for each disk $D_k \in D_i$ and $D_l \in D_j$ such that $d(p_i, p_k)$ and $d(p_j, p_l)$ are minimized. In other words, $p_k$ and $p_l$ correspond to the first intersections respectively between $\overrightarrow{p_i c_k}$ and $D_k$ and between $\overrightarrow{p_j c_l}$ and $D_l$.

The idea of our algorithm is to compute a minimal placement for every pair of polar disks $D_i$ and $D_j$ and every essential 2-partition $D_i, D_j$.
of $\mathcal{D} \setminus \{D_i, D_j\}$ and select the one resulting the minimum diameter (we define an “essential 2-partition” in the following). We use the arrangement generated by the bisectors among disks to show that, for a fixed pair of polar disks, the number of essential 2-partitions to consider is $O(n^5)$. We then use $\text{FV}(\mathcal{D}_i)$ and $\text{FV}(\mathcal{D}_j)$ to prove that there exists a minimum placement where $p_i$ and $p_j$ lie on the boundaries of $D_i$ and $D_j$, respectively, and to compute the optimal locations in $O(n^9)$ time, for an overall run-time of $O(n^{10})$. These results rely on the following fact, derived from the “stability condition” of Ho et al. [70] stating that the longest path of a bipolar minimum diameter spanning tree contains the edge between the two poles.

**Fact 6.10.** For a bipolar minimum diameter spanning tree with poles $p_i \in D_i$ and $p_j \in D_j$ it holds that

$$\delta_{ij} = d(p_i, D_k) + |p_ip_j| + d(p_j, D_l),$$

where $D_k$ and $D_l$ are respectively the farthest disks of $p_i$ in $\mathcal{D}_i$ and of $p_j$ in $\mathcal{D}_j$. Thus, $p_i$ belongs to $\text{FVR}(D_k, \mathcal{D}_i)$ and $p_j$ belongs to $\text{FVR}(D_l, \mathcal{D}_j)$.

**Bipolar topologies.** To bound the number of bipolar tree topologies we look at the situation where the poles $p_i, p_j$ and the 2-partition $\mathcal{D}_i, \mathcal{D}_j$ are fixed. If $D_k$ is the farthest disk of $p_i$ in $\mathcal{D}_i$, by Fact 6.10 moving any disk $D \in \mathcal{D}_j$ with $d(p_i, D) \leq d(p_i, D_k)$ into $\mathcal{D}_i$ will not increase $\delta_{ij}$. It is thus sufficient to consider only those topologies where $\mathcal{D}_i$ contains every disk in $\mathcal{D} \setminus \{D_i, D_j\}$ whose distance to $p_i$ is at most $d(p_i, D_k)$ and $\mathcal{D}_j$ contains the remaining disks.

More formally, we define a sequence of disks $(B_1, B_2, \ldots, B_{n-2})$ to be the nearest ordered sequence of a pole $p_i \in D_i$ if $B_m$ is the $m^{th}$ nearest disk of $p_i$ in $\mathcal{D} \setminus \{D_i, D_j\}$ for $1 \leq m \leq n - 2$ (with ties broken arbitrarily). If $B_m$ is the farthest disk of $p_i$ in $\mathcal{D}_i$, replacing $\mathcal{D}_i$ with $\{B_1, \ldots, B_m\}$ and $\mathcal{D}_j$ with $\{B_{m+1}, \ldots, B_{n-2}\}$ (and moving the points in the placement accordingly) does not increase $\delta_{ij}$. By this argument, once the pole $p_i$ is fixed (even regardless of $p_j$), the number of different candidates for $\mathcal{D}_i$ and $\mathcal{D}_j$ that need to be considered is $n - 2$, corresponding to $\mathcal{D}_i = \{B_1, \ldots, B_m\}$ and $\mathcal{D}_j = \{B_{m+1}, \ldots, B_{n-2}\}$ for $1 \leq m \leq n - 2$. As a result, for a pair of polar disks $D_i$ and $D_j$, if we could bound the total number of different nearest ordered sequences of $\mathcal{D} \setminus \{D_i, D_j\}$ for all points $p_i \in D_i$ by a polynomial, the number of bipolar tree topologies we would need to consider is therefore equal to the polynomial times $(n - 2)$. This bound
Figure 6.6: Arrangement formed by bisectors among points and the resulting nearest ordered sequence.

is proven in the following lemma using the arrangement formed by bisectors among the disks in $D \{D_i, D_j\}$.

**Lemma 6.11.** For a pair of polar disks $D_i$ and $D_j$ it is sufficient to consider $O(n^5)$ 2-partitions that can be enumerated in $O(n^5)$ time.

**Proof.** We consider the arrangement formed by the $\binom{n-2}{2}$ bisectors between all pairs of disks in $D \{D_i, D_j\}$ (see Figure 6.6 for an example on points). For the first part of the statement, observe that all points in a face of the arrangement share the same nearest ordered sequence of $D \{D_i, D_j\}$, because there is no bisector passing through it. Since any two bisectors intersect at most four times (even for amended bisectors), the arrangement has $O(n^4)$ faces. Therefore, the total number of different nearest ordered sequences of $D \{D_i, D_j\}$ for all points in the plane is $O(n^4)$, and the number of 2-partitions that need to be considered is $O(n^5)$.

For the second part of the statement, we show how to construct the arrangement in $O(n^5)$ time by incrementally inserting disks. When
inserting the $k^{th}$ disk, we consider the $k - 1$ bisectors between it and the previous $k - 1$ disks. Each new bisector makes $O((k - 1)^2)$ intersections with the previous $\binom{k-1}{2}$ ones, leading to $O((k - 1)^3)$ intersections in total. Therefore, the overall time to compute the arrangement is $\sum_{2 \leq k \leq n-2} O((k - 1)^3) = O(n^4)$. For every face intersecting $D_i$ we generate the corresponding nearest ordered sequence and the $n - 2$ candidates for $O_i$, resulting in a total run-time of $O(n^5)$.

**Locations of poles.** For a fixed bipolar tree topology consisting of $D_i, D_j$ and $O_i, O_j$, the following lemma employs $FV(O_i)$ and $FV(O_j)$ to analyze the optimal locations of the poles $p_i$ and $p_j$.

**Lemma 6.12.** If $D$ does not admit monopolarity, there exists a minimum placement and a bipolar minimum diameter spanning tree with poles $p_i \in D_i$ and $p_j \in D_j$ such that $p_i \in \partial D_i$ and $p_j \in \partial D_j$.

**Proof.** Without loss of generality, we assume $p_i$ and $p_j$ to not belong to $D_i \cap D_j$, since otherwise placing $p_i$ on $p_j$ would result in a minimum placement admitting a monopolar minimum diameter spanning tree. If $p_i \notin \partial D_i$, we move $p_i$ along the direction $\overrightarrow{p_i p_j}$ until we reach $\partial D_i$. While moving $p_i$, even if its farthest disk $D_k$ in $O_i$ changes, $d(p_i, D_k) + |\overrightarrow{p_ip_j}|$ will not increase, and thus the diameter will not increase. If also $p_j \notin \partial D_j$, we move $p_j$ to $\partial D_j$ in a symmetric way without increasing the diameter, leading to this lemma.

We compute a minimal placement for a bipolar tree topology $D_i, D_j$, $O_i, O_j$ as follows. First, we compute $FV(O_i)$ and $FV(O_j)$ and use them to partition $\partial D_i$ and $\partial D_j$ into arcs. For each pair of such arcs $a_k \in \partial D_i \setminus FV(O_i)$ and $a_l \in \partial D_j \setminus FV(O_j)$, we then find the points $p'_i \in cl a_k$ and $p'_j \in cl a_l$ minimizing the corresponding diameter. Finally, we select $p_i$ and $p_j$ as the pair $p'_i$ and $p'_j$ resulting in the smallest diameter.

**Theorem 6.13.** If $D$ does not admit monopolarity, a minimum placement for $D$ can be computed in $O(n^9)$ time.

**Proof.** By Lemma 6.11, for each pair of polar disks $D_i, D_j$ we consider $O(n^5)$ 2-partitions of $D \setminus \{D_i, D_j\}$. For each 2-partition $O_i, O_j$ we compute $FV(O_i)$ and $FV(O_j)$ by Theorem 6.6 in $O(n \log^2 n)$ time. Since by Theorem 6.5 both diagrams have $O(n)$ edges, both $\partial D_i \setminus FV(O_i)$ and $\partial D_j \setminus FV(O_j)$ have $O(n)$ arcs and $O(n^2)$ pairs of arcs will be dealt with.
We process each pair of arcs by Fact 6.4 in $O(1)$ time and thus compute a minimum placement for fixed $D_i, D_j$ and $\mathcal{D}_i, \mathcal{D}_j$ in $O(n^2)$ time, for a total run-time of $O(n^9)$.

**Disjoint Disks**

Since the run-time is dominated by the bipolar case, we now look at how to improve the computation for this particular case. For pair-wise disjoint disks, we propose a linear-time sequential search to compute a minimum placement for a fixed topology $D_i, D_j, \mathcal{D}_i, \mathcal{D}_j$, leading to $O(n^8 \log^2 n)$ run-time.

For this purpose, we reformulate $\delta_{ij}$ as a function of $p_i \in \partial D_i$ and $p_j \in \partial D_j$ parametrizing $p_i$ and $p_j$ by two angles $\theta_i$ and $\theta_j$ such that $p_i = (r_i \cos \theta_i + h_i, r_i \sin \theta_i + k_i)$ and $p_j = (r_j \cos \theta_j + h_j, r_j \sin \theta_j + k_j)$. Since $\delta_{ij}$ is attained by a path from $p_k$ to $p_l$ passing through $p_i$ and $p_j$, for some $D_k \in \mathcal{D}_i$ and $D_l \in \mathcal{D}_j$, we define a function

\[
f_{k,l}(\theta_i, \theta_j) = d(p_i, D_k) + |p_i p_j| + d(p_j, D_l) = |c_k p_i| + |p_i p_j| + |p_j c_l| - r_k - r_l
\]

representing the length of such a simple path. Regardless of the locations of $p_i$ and $p_j$, we can then express $\delta_{ij}$ as

\[
f(\theta_i, \theta_j) = \max_{D_k \in \mathcal{D}_i, D_l \in \mathcal{D}_j} f_{k,l}(\theta_i, \theta_j).
\]

The following lemmas indicate that a local minimal point of $f_{k,l}$ is a global minimum one, and that the same property also holds for $f$.

**Lemma 6.14.** If $f_{k,l}$ has a local minimum point at $\theta_i$ and $\theta_j$, the point is also a global minimum.

**Proof.** We observe that, as shown by Figure 6.7, the path connecting $p_k$ and $p_l$ and passing through $p_i$ and $p_j$ can bend at most 2 times. We therefore distinguish three possible cases according to the number of bends of this path.

In case the path makes no bend as in Figure 6.7a, local minimal points only occur when $p_i \in c_k c_j \cap \partial D_i$ and $p_j \in c_k c_j \cap \partial D_j$ since $c_k, p_i, p_j$ and $c_l$ are on the same line. It is clear that all those points achieve the same value of $f_{k,l}$. 
If the path makes only one bend as in Figure 6.7b, then either \( p_i \in \overrightarrow{c_kp_j} \cap \partial D_i \) or \( p_j \in \overrightarrow{p_lc_i} \cap \partial D_j \). For the former there are exactly two local minimal points, occurring when \( p_j \) minimizes \( |\overrightarrow{c_kp_j}| + |\overrightarrow{p_jc_l}| \) and \( p_i \) is one of the two intersections between \( \overrightarrow{c_kp_j} \) and \( \partial D_i \). In other words, \( \overrightarrow{c_ip_j} \) is an angular bisector of \( \angle c_kp_jc_l \). The two minimal points clearly achieve the same value of \( f_{k,l} \). A symmetric argument holds for the latter.

For the remaining case where the path makes two bends as in Figure 6.7c there is exactly one local minimal point, occurring when \( \overrightarrow{c_ip_j} \) bisects \( \angle c_kp_ip_j \) and \( \overrightarrow{c_jp_j} \) bisects \( \angle p_jp_lc_l \).

**Lemma 6.15.** If \( f \) has a local minimum point at \( \theta_i \) and \( \theta_j \), the point is also a global minimum.

**Proof.** Assume that \( f \) has a global minimum point at \( \theta'_i \) and \( \theta'_j \). Let \( D_k \) and \( D_{k'} \) be the farthest disks respectively of \( p_i \) and \( p'_i \) in \( \mathcal{D}_i \), and \( D_l \) and \( D_{l'} \) be the farthest disks respectively of \( p_j \) and \( p'_j \) in \( \mathcal{D}_j \). By Lemma 6.14, the definition of \( f \), and the fact that \( f \) has a global minimum point at \( \theta'_i \) and \( \theta'_j \), it holds that

\[
 f_{k,l}(\theta_i, \theta_j) \leq f_{k,l}(\theta'_i, \theta'_j) \leq f_{k',l'}(\theta'_i, \theta'_j) \leq f_{k,l}(\theta_i, \theta_j). 
\]
Chapter 6. Spanning Trees

Thus, \( f_{k,l}(\theta_i, \theta_j) = f_{k',l'}(\theta'_i, \theta'_j) \) and \( f \) has a global minimum point at \( \theta_i, \theta_j \).

Corollary 6.16. If \( f \) has a local minimum point at \( \theta_i \) for fixed \( \theta_j \) (resp. at \( \theta_j \) for fixed \( \theta_i \)), the point is also a global minimum for fixed \( \theta_j \) (resp. for fixed \( \theta_i \)).

In order to present the sequential search, we re-interpret \( f \) geometrically as shown by Figure 6.8. We view \( (\theta, \phi, f_{k,l}(\theta, \phi)) \) as a point in three dimension, so that \( f_{k,l} \) is a 3D surface and \( f \) is the upper envelope of all \( f_{k,l} \). If we assign different colors to different \( f_{k,l} \), the vertical projection of \( f \) onto the \( xy \)-plane is a planar subdivision consisting of axis-parallel rectangles, where each rectangle is associated with a function \( f_{k,l} \) such that \( f_{k,l}(\theta_i, \theta_j) = f(\theta_i, \theta_j) \) if \( (\theta_i, \theta_j) \) belongs to the rectangle. Therefore, a point \( (\theta_i, \theta_j) \) in a rectangle associated with \( f_{k,l} \) corresponds to a point \( p_i \) on the arc \( a_k = \text{FVR}(D_k, \partial D_i) \cap \partial D_i \) and a point \( p_j \) on the arc \( a_l = \text{FVR}(D_l, \partial D_j) \cap \partial D_j \). Searching the minimal point of \( f_{k,l} \) inside the corresponding rectangle is equivalent to searching the minimum \( \delta_{ij} \) for a pair of poles \( p_i \) and \( p_j \) on the corresponding arcs.

We can sequentially search the rectangles until we find a point \( (\theta_i, \theta_j) \) with minimum \( f(\theta_i, \theta_j) \). We begin with an arbitrary rectangle and compute the minimal value inside it, i.e., the minimum \( \delta_{ij} \) achieved by a pair of points on the corresponding arcs. If the minimal point occurs in the interior of the rectangle, by Lemma 6.15 we have found a global minimum point. Otherwise, we compute the minimum values for the at most three other rectangles adjacent to the minimal point and select the smallest one. If the smallest value of the selected adjacent rectangle is not smaller than that of the previous one, then the previous
point was a global minimum. Otherwise, we continue searching in the selected rectangle. The following lemma indicates the run-time of this sequential search. Note that the geometric interpretation is presented only for easier understanding; we do not construct the corresponding planar subdivision due to its quadratic size.

**Lemma 6.17.** For fixed $D_i, D_j, \mathcal{D}_i, \mathcal{D}_j$, a pair of poles $p_i \in \partial D_i$ and $p_j \in \partial D_j$ with minimum $\delta_{ij}$ can be computed in $O(|\partial D_i \setminus \text{FV}(\mathcal{D}_i)| + |\partial D_j \setminus \text{FV}(\mathcal{D}_j)|)$ time.

*Proof.* Moving from one rectangle to the next one, at least one of the corresponding arcs is replaced with its adjacent arc. By Corollary 6.16, if we leave one arc it will not be considered again. Therefore, the number of pairs of tested arcs is $O(|\partial D_i \setminus \text{FV}(\mathcal{D}_i)| + |\partial D_j \setminus \text{FV}(\mathcal{D}_j)|)$. By Fact 6.4 a solution for a pair of arcs can be computed in $O(1)$ time, proving the statement. □

Since $O(|\partial D_i \setminus \text{FV}(\mathcal{D}_i)| + |\partial D_j \setminus \text{FV}(\mathcal{D}_j)|) = O(n)$, the time for computing a minimum placement for fixed $D_i, D_j, \mathcal{D}_i, \mathcal{D}_j$ is bounded by the $O(n \log^2 n)$ construction time of $\text{FV}(\mathcal{D}_i)$ and $\text{FV}(\mathcal{D}_j)$. There are $O(n^2)$ pairs of disks and we consider $O(n^5)$ topologies per pair, resulting in the following overall run-time.

**Theorem 6.18.** If all disks are pairwise disjoint, a minimum placement for $\mathcal{D}$ can be computed in $O(n^8 \log^2 n)$ time.

**Unit Disks**

If all disks are unit, we further improve the run-time to $O(n^6)$ for the intersecting case and to $O(n^5)$ for the disjoint case. We obtain these improvements by reducing the number of bipolar topologies that need to be considered and decreasing the overall time required to compute all the Voronoi diagrams.

**Separable topologies.** The key property to reduce the number of bipolar tree topologies to consider is proven by the following lemma.

**Lemma 6.19.** If all disks in $\mathcal{D}$ are unit, there exists a minimum placement that admits:
112 Chapter 6. Spanning Trees

Figure 6.9: The cases of Lemma 6.19.

1. a bipolar minimum diameter spanning tree with poles \( p_i \in D_i \) and \( p_j \in D_j \) such that the centers of \( D_i \) and the centers of \( D_j \) can be separated by a straight line, or

2. a monopolar minimum diameter spanning tree.

Proof. Consider a minimum placement with a bipolar minimum diameter spanning tree whose poles are \( p_i \in D_i \) and \( p_j \in D_j \). We modify \( D_i \) and \( D_j \) to satisfy the statement without increasing the diameter. Let \( B_i \) and \( B_j \) denote the smallest balls centered respectively at \( p_i \) and \( p_j \) and enclosing respectively the centers of \( D_i \) and \( D_j \). Let \( R_i \) and \( R_j \) be the radii of \( B_i \) and \( B_j \), respectively. If \( R_i < 1 \), all disks in \( D_i \) contain \( p_i \) and all points for \( D_i \) are placed on \( p_i \), so the distance between \( p_i \) and its farthest disk in \( D_i \) is 0. Otherwise, if \( R_i \geq 1 \), such a distance is \( R_i - 1 \). By symmetry for \( R_j \) and Fact 6.10, it thus holds \( \delta_{ij} = \max\{R_i - 1, 0\} + \max\{R_j - 1, 0\} + |p_i - p_j| \). If \( B_i \) and \( B_j \) do not intersect or are tangent (see for example Figure 6.9a), the first statement trivially holds. Otherwise, we distinguish the relation between \( B_i \) and \( B_j \) into two cases.

If \( B_j \subset B_i \) as in Figure 6.9b, we replace \( p_l \in D_l \) for all \( D_l \in D_j \) with
the first intersection between \( p_i \) and \( D_l \) and link \( p_l \) to \( p_i \) instead of \( p_j \). Since these operations result in a monopolar spanning tree and do not increase the diameter, the second statement holds. The symmetric holds if \( B_i \subset B_j \).

If \( B_i \cap B_j \neq \emptyset \) but neither ball is completely contained inside the other as in Figure 6.9c, we construct a minimum placement admitting a bipolar minimum diameter spanning tree satisfying the first statement as follows. Let \( L \) be the line passing through the two intersections between \( \partial B_i \) and \( \partial B_j \) and \( H_i \) and \( H_j \) be the two connected regions of \( \mathbb{R}^2 \) separated by \( L \). We move each \( D_k \in D_i \) whose center \( c_k \) belongs to \( H_j \) from \( D_i \) to \( D_j \) and each \( D_l \in D_j \) whose center \( c_l \) belongs to \( H_i \) from \( D_j \) to \( D_i \). Since the centers of the moved disks belong to \( B_i \cap B_j \), the movement will not increase neither \( R_i \) nor \( R_j \), leading the first statement.

An essential 2-partition of \( D \setminus \{D_i, D_j\} \) thus corresponds to a 2-partition of the centers \( C \setminus \{c_i, c_j\} \) that can be separated by a straight line. The following well-known lemma implies that, for a pair of polar disks \( D_i \) and \( D_j \), the number of 2-partitions to consider is \( O(n^2) \), and they can be computed in \( O(n^2) \) time.

**Lemma 6.20.** For a set \( S \) of \( n \) points, there are \( O(n^2) \) 2-partitions of \( S \) into \( S_1 \) and \( S_2 \) such that there exists a line separating \( S_1 \) and \( S_2 \), and those 2-partitions can be generated in \( O(n^2) \) time.

**Proof.** The statement is well-known and we include the proof for the sake of completeness. We make use of the central point-line geometry duality \( \Psi \). For a point \( s = (h, k) \) different from the origin, \( \Psi(s) \) is the line \( hx + ky = 1 \), and for a line \( L: \alpha x + \beta y = 1 \) which does not pass through the origin, \( \Psi(L) \) is the point \((\alpha, \beta)\). A point \( s \) and the origin are on the same side of a line \( L \) if and only if \( \Psi(L) \) and the origin are on the same side of \( \Psi(s) \). Therefore, the lines in \( \Psi(S) = \{\Psi(s) \mid s \in S\} \) form an arrangement such that, for all points \( t \) and \( t' \) in the same cell, \( \Psi(t) \) and \( \Psi(t') \) result in the same 2-partition of \( S \).

For two cells separated by \( \Psi(s) \), the corresponding 2-partitions differ only by \( s \). A depth-first search in the arrangement forms a tour visiting all the 2-partitions of length \( O(n^2) \). It is well-known [33] that the arrangement of \( n \) lines can be computed in \( O(n^2) \) time, and thus the tour can be generated in \( O(n^2) \) time.
Computing Voronoi Diagrams in a batch. To further improve the run-time, for a fixed pair of polar disks $D_i, D_j$ we compute all $O(n^2)$ diagrams $FV(D_i)$ and $FV(D_j)$ in a batch. Since for unit disks it holds that $FV(D) = FV(C)$, we can use two well-known facts: the farthest-site Voronoi diagram of $n$ point sites can be computed in $O(n)$ time if their convex hull is pre-computed [3], and we can dynamically update convex hulls in $O(\log^2 n)$ time [112].

**Lemma 6.21.** If $D$ are all unit disks, for a pair of polar disks $D_i$ and $D_j$ all the $O(n^2)$ 2-partitions of $D \setminus \{D_i, D_j\}$ satisfying Lemma 6.19 and the corresponding farthest-disk Voronoi diagrams can be computed in $O(n^3)$ time.

**Proof.** By Lemma 6.20, there exists a tour visiting all $O(n^2)$ 2-partitions of $D \setminus \{D_i, D_j\}$ satisfying Lemma 6.19 such that two consecutive 2-partitions along the tour differ by only one disk. Furthermore, the length of such a tour is $O(n^2)$ and it can be computed in $O(n^2)$ time. For computing the $O(n^2)$ farthest-disk Voronoi diagrams of all the necessary 2-partitions, we begin by constructing the convex hulls of the two sets of centers in the first 2-partition along the tour. When moving from a 2-partition to the next one along the tour, we update the convex hulls by inserting one center point into one hull and deleting it from the other one. We adopt Overmars and van Leeuwen’s dynamic structure [112] to maintain the convex hulls, which performs both an insertion and a deletion operation in $O(\log^2 n)$ time. Since we perform $O(n^2)$ insertions and deletions, it takes $O(n^2 \log^2 n)$ time in total to update the two convex hulls along the tour. When we visit a 2-partition, we compute the farthest-site Voronoi diagrams from the respective convex hulls in $O(n)$ time using the algorithm by Aggarwal et al. [3]. Since there are $O(n^2)$ 2-partitions along the tour, the time to compute the farthest-disk Voronoi diagrams of all the necessary 2-partitions of $D \setminus \{D_i, D_j\}$ is $O(n^3)$. 

For a pair of polar disks $D_i$ and $D_j$ we find a minimum placement by computing all the $O(n^2)$ 2-partitions of $D \setminus \{D_i, D_j\}$ together with the corresponding farthest Voronoi diagrams in $O(n^3)$ time. For each 2-partition, a minimum placement can be found in $O(n^2)$ and $O(n)$ time respectively for intersecting and for disjoint disks, leading to $O(n^3)$ and

---

1 An improved $O(\log n)$ bound is known for the time required to update a convex hull [23]. The conference paper does not however explain in sufficient detail all the steps of the algorithm, and a journal version has not been published yet. We point out that, even with the improved bound, the worst-case run-time of our algorithms do not change.
Implementation of Elementary Operations

115

Theorem 6.22. If all disks in \( \mathcal{D} \) are unit, a minimum placement can be computed in \( O(n^6) \) time and \( O(n^5) \) time respectively for intersecting and for disjoint disks.

6.3 Implementation of Elementary Operations

In the following we present the implementation of the elementary operations of Fact 6.3 and Fact 6.4. These implementations are presented for the sake of completeness; a reader interested only in results concerning optimization for imprecise points can safely skip this section.

Fact 6.3. We distinguish cases according to the location of \( t \in e \):

1. \( t \in D_p \cap D_q \): Any point \( t \in e \cap (D_p \cap D_q) \) satisfies \( d(t, D_p) + d(t, D_q) = 0 \), and \( e \cap (D_p \cap D_q) \) can be computed in \( O(1) \) time.

2. \( t \in D_p \setminus D_q \) (resp. \( t \in D_q \setminus D_p \)): \( d(t, D_p) + d(t, D_q) = d(t, D_q) = |tc_q| - r_q \) (resp. \( d(t, D_p) + d(t, D_q) = d(t, D_p) = |tc_p| - r_p \)), and thus the location of \( t \) minimizing \( d(t, D_p) + d(t, D_q) \) can be trivially computed in \( O(1) \) time.

3. \( t \notin D_p \cup D_q \): \( d(t, D_p) + d(t, D_q) = |tc_p| + |tc_q| - r_p - r_q \). We discuss below how to compute \( t \) minimizing \( d(t, D_p) + d(t, D_q) \) for this case in \( O(1) \) time.

For the sake of simplicity, we use \( p \) to denote \( c_p \), \( q \) to denote \( c_q \) and, since \( r_p \) and \( r_q \) are fixed, we attempt to minimize \( |pt| + |qt| \). Such a point \( t \in e \) can be located either in the interior of \( e \) or at one of its endpoints. In the latter case we can easily find the endpoint of \( e \) resulting in the smallest value by checking both of them. In the former case, if \( e \) and the segment \( pq \) intersect, then \( t \) is trivially the intersection point. Otherwise we distinguish between the cases where \( e \) is a line segment, a circular arc, or a hyperbolic arc.

If \( e \) is a line segment, it is well-known that the line perpendicular to \( e \) and passing through \( t \) bisects the angle \( \angle ptq \) (Figure 6.10a). This point can be computed in \( O(1) \) time as the intersection between \( \overline{pq} \) and \( e \),
where \( p' \) is the reflection point of \( p \) with respect to the line passing through \( e \).

If \( e \) is a circular arc, the line passing through \( t \) and the center \( c \) of the circle induced by \( e \) bisects the angle \( \angle ptq \) (Figure 6.10b). To compute this point we adopt the Newton-Raphson method. In order to do so, we formulate a function \( f(z) \) such that \( f(z) = 0 \) if and only if a ray shot from \( p \) in the direction \( (1, z) \) first hits \( e \), then reflects on \( e \), and finally passes through \( q \). In other words, if \( f(z) = 0 \), then \( t \) is the first intersection between \( e \) and the ray shot from \( p \) along the direction \( (1, z) \).

For simplicity, assume that \( e \) induces a unit circle \( C \) with center \( c = (h, k) \). Let \( (x_p + \tau, y_p + z\tau), \tau \geq 0 \) be the ray shooting from \( p = (x_p, y_p) \) in the direction \( (1, z) \). This ray hits \( C \) at the point \( (x_p + f_1(z), y_p + f_1(z)z) \) such that

\[
a = x_p - h, \quad b = y_p - k,
\]

\[
f_1(z) = \frac{-2(2a + 2bz) - \sqrt{4(1 - a^2)z^2 + 8abz + (1 - b^2)}}{2(z^2 + 1)}.
\]

After hitting \( C \) the ray reflects at \( (x_p + f_1(z), y_p + f_1(z)z) \) and continues in the direction \( (x_p + f_1(z) + \tau, y_p + f_1(z)z + f_3(z)\tau), \tau \geq 0 \), where

\[
f_2(z) = \frac{2(b - az)f_1(z)}{(b + f_1(z)z)^2 + (a + f_1(z))^2},
\]

\[
f_3(z) = \frac{-(a + f_1(z))f_2(z) - f_1(z)z}{(b + f_1(z)z)f_2(z) - f_1(z)}.
\]
Since the ray will pass through \( q = (x_q, y_q) \), we have \((y_q - y_p) - f_1(z)(z - f_3(z)) - f_3(z)(x_q - x_p) = 0\). Thus, we can set \( f \) as

\[
f(z) = (y_q - y_p) - f_1(z)(z - f_3(z)) - f_3(z)(x_q - x_p).
\]

To apply the Newton-Raphson method we need an interval \([z_l, z_r]\) containing the root \( z_0 \) of \( f(z) \) and such that for every \( z \in [z_l, z_0) \) and \( z \in (z_0, z_r] \) it holds \( f'(z) \neq 0 \). We can pick as \( z_l \) and \( z_r \) two distinct values such that the rays shot from \( p \) along the directions \((1, z_l)\) and \((1, z_r)\) are tangent to \( C \).

For the analysis of the run-time, since all the sub-functions of \( f \) have a second derivative in the interval \([z_l, z_r]\), \( f \) also has a second derivative in \([z_l, z_r]\). It is well-known if \( f \) has a second derivative in \([z_l, z_r]\), the convergence rate of the Newton-Raphson method is quadratic in \([z_l, z_r]\). According to the time complexity analysis for the Newton-Raphson method by Borwein and Borwein [22], if the convergence rate is quadratic the root \( z_0 \) can be found in \( O(M(m)) \) bit-operations, where \( M(m) \) is the number of bit-operations required to multiply two \( m \)-bits words. By the standard assumption of using the RAM model of computation, the multiplication between two numbers is considered a constant-time operation. The time required to find a point \( t \) on a circular arc \( e \) minimizing \( |\overrightarrow{pt}| + |\overrightarrow{qt}| \) is therefore also in \( O(1) \).

In the case where \( e \) is a hyperbolic arc, a reflection property similar to the circular arc case holds. The above method can be generalized in a straightforward fashion for hyperbolic arcs.

**Fact 6.4.** We denote the circles induced by \( e_1 \) and \( e_2 \) respectively as \( C_1 \) and \( C_2 \), their centers respectively as \( c_1 = (h_1, k_1) \) and \( c_2 = (h_2, k_2) \), and their radii respectively as \( r_1 \) and \( r_2 \). We distinguish four cases depending on the locations of \( s \in e_1 \) and \( t \in e_2 \).

1. \( s \in D_p \) and \( t \in D_q \): \( d(s, D_p) + |\overrightarrow{st}| + d(t, D_q) = |\overrightarrow{st}| \), and the locations of \( s \) and \( t \) to minimize \( |\overrightarrow{st}| \) can be trivially computed in \( O(1) \) time.

2. \( s \in D_p \) but \( t \not\in D_q \): \( d(s, D_p) + |\overrightarrow{st}| + d(t, D_q) = |\overrightarrow{st}| + d(t, D_q) \), and the locations of \( s \) and \( t \) to minimize \( |\overrightarrow{st}| + d(t, D_q) \) can be computed in \( O(1) \) time by Fact 6.3 where \( s \) is replaced with \( C_1 \).

3. \( t \in D_q \) but \( s \not\in D_p \): It is symmetric to the second case.
Chapter 6. Spanning Trees

Figure 6.11: Reflection on two circular arcs.

4. $s \notin D_p$ and $t \notin D_q$: $d(s, D_p) + |st| + d(t, D_q) = |cp| + |st| + |tc_q|$

We discuss below how to compute $s, t$ minimizing $d(t, D_p) + |st| + d(t, D_q)$ for this case in $O(1)$ time.

We use $p$ to denote $c_p$, $q$ to denote $c_q$, and assume that $C_1$ and $C_2$ are unit circles. We distinguish three possibilities for the locations of $s \in \text{cl}_1$ and $t \in \text{cl}_2$:

1. Both $s$ and $t$ are endpoints of their corresponding arcs: We can then check all 4 possibilities and find the one resulting in the smallest value.

2. Either $s$ or $t$ is is an endpoint of its corresponding arc: We can then apply the Newton-Raphson method presented above for each endpoint and find the one resulting in the smallest value. For example, if $s$ is an endpoint of $e_1$, we find a point $t \in C_2$ minimizing $|st| + |tq|$

3. Both $s$ and $t$ lie in the interior of their corresponding arcs: We propose the method below to compute their locations in $O(1)$ time.

We distinguish between the cases where the segment $pq$ intersects both $C_1$ and $C_2$, where it intersects only one among $C_1$ and $C_2$, and where it intersects neither $C_1$ nor $C_2$.

If $pq$ intersects both $C_1$ and $C_2$ the points $s$ and $t$ can be computed by setting $s$ as one of the two intersections between $pq$ and $C_1$ and $t$ as one of the two intersections between $pq$ and $C_2$.

If $pq$ intersects neither $C_1$ nor $C_2$ then the line passing through $c_1$ and $s$ will bisect $\angle qst$ and the line passing through $c_2$ and $t$ will bisect...
\(\triangle stq\) (Figure 6.11). We design a Newton-Raphson method algorithm similar to the one above to compute \(s\) and \(t\) in \(O(1)\) time. Again, we formulate a function \(f(z)\) such that \(f(z) = 0\) if and only if a ray shot from \(p = (x_p, y_p)\) in the direction \((1, z)\) first reflects on \(C_1\), then reflects on \(C_2\), and finally passes through \(q = (x_q, y_q)\). In other words, if \(f(z) = 0\), then \(s\) is the first intersection between \(e_1\) and the ray shot from \(p\) along the direction \((1, z)\), and \(t\) is the first intersection between \(e_2\) and the ray reflecting at \(s\). The function \(f\) can be defined in a similar way as in the previous case:

\[
f(z) = ((f_3(z) + f_6(z))f_7(z) + f_6(z)f_5(z))x_q + \\
( (f_4(z) + f_6(z)f_5(z))f_7(z) - f_6(z))y_q - \\
( (f_4(z) + f_6(z)f_5(z))f_7(z) - f_6(z))(y_p + f_1(z)z + f_6(z)f_5(z)) - \\
( (f_3(z) + f_6(z))f_7(z) + f_6(z)f_5(z))(x_p + f_1(z) + f_6(z)),
\]

where

\[
a = x_p - h_1, \ b = y_p - k_1,
\]

\[
f_1(z) = \frac{-(2a + 2bz) - \sqrt{4(1 - a^2)z^2 + 8az + (1 - b^2)}}{2(z^2 + 1)},
\]

\[
f_2(z) = \frac{2(b - az)f_1(z)}{(b + f_1(z)z)^2 + (a + f_1(z))^2},
\]

\[
f_3(z) = (x_p - h_2) + f_1(z),
\]

\[
f_4(z) = (y_p - k_2) + f_1(z)z,
\]

\[
f_5(z) = \frac{-(a + f_1(z))f_2(z) - f_1(z)z}{(b + f_1(z)z)f_2(z) - f_1(z)},
\]

\[
f_6(z) = \frac{-(2f_3(z)+2f_4(z)f_5(z)) - \sqrt{4(1-f_3(z)^2)z^2+8f_3(z)f_4(z)f_5(z)+(1-f_4(z))^2}}{2(f_3(z)^2+1)},
\]

\[
f_7(z) = \frac{2(f_4(z) - f_3(z)f_5(z))f_6(z)}{(f_4(z) + f_6(z)f_5(z))^2 + (f_3(z) + f_6(z))^2}.
\]

The Newton-Raphson method requires an interval \([z_l, z_r]\) containing the root \(z_0\) of \(f\) such that for every \(z \in [z_l, z_r]\) and \(z \in (z_0, z_r), f'(z) \neq 0\). We can pick two distinct values as \(z_l\) and \(z_r\) such that the rays shot from \(p\) along the directions \((1, z_l)\) and \((1, z_r)\) will first reflect at \(C_1\) and then be tangent to \(C_2\). Since also in this case all the sub-functions of \(f\) have a second derivative in \([z_l, z_r]\), the Newton-Raphson method converges to the root \(z_0\) in \(O(1)\) time.
For the last case, assume without loss of generality that $pq$ intersects only $C_1$. If $pt$ passes through $C_1$, then we can set $t$ as any of the two intersection between $pt$ and $C_1$, while the point $s$ can be computed using the Newton-Raphson method of Fact 6.3. If $pt$ does not pass through $C_1$, we can instead use the above Newton-Raphson method.
Concluding Remarks

We investigated algorithmic problems in the context of optimization under uncertainty for applications related to routing and spanning trees. Our results range from theoretical proofs and algorithms to practically efficient implementations. Even though we were able to answer some of our open questions, the work on these topics is not finished.

In particular, the run-time of the enumeration algorithm used to compute optimum paths according to the maximum similarity criterion is in the worst-case exponential. This algorithm was employed for a simple preliminary investigation and it is clearly of no practical use. However, since the method appears to produce routes of good quality, practical implementations are of high interest. For the computation of paths according to the first intersection criterion, we engineered a bi-directional algorithm. The performance of this algorithm on real-world instances is quite good, but its run-time is still not fast enough for practical applications. We believe though that there is still room for improvement. A possible direction for further study is the application of existing techniques for speeding-up the computation of shortest paths.

We point out that the mixing time of the Markov chain $M_{\text{paths}}$ to sample simple paths in planar graphs is not known for $\lambda \in (0, 1)$. Our conjecture is that, also in this case, the chain is not rapidly mixing. We note however that the graphs with high bottleneck ratio used in the proofs Theorem 3.7 and corollary 3.8 cannot be applied to this case. A further object of study is, of course, the design of an efficient sampler for simple paths in general and/or planar graphs.

The computational complexity of some variants of the shortest path problem for imprecise points remains open. For example, we have
shown the MIN-SPP to be solvable in polynomial time for axis-aligned rectilinear polygons if distances are measured with the $L_1$ metric. It is not known whether the same claim can be made for the same kind of polygons if distances are measured with some other metric $L_p$ with $p > 1$. For the maximization variant, we have shown that the MAX-SPP cannot be approximated within any factor $(1 - \epsilon)$ with $\epsilon < 1/4$. It is unknown whether the problem admits a polynomial-time algorithm with approximation factor of $3/4$.

The literature on minimum weight spanning trees for imprecise points indicates some open directions. The problem of computing a smallest minimum weight spanning tree has been shown to not admit an FPTAS in the case where the uncertainty regions are in the shape of axis-aligned segments [40] or disks [42]. The presence of a PTAS has been answered in the positive for the latter case [130] but remains open in the former. For the computation of a largest minimum weight spanning tree, the existence of a PTAS is open both for polygonal and for disks-shaped regions.

To conclude, we looked at the computation of smallest minimum diameter spanning trees for imprecise points for disk-shaped regions. We provided polynomial time algorithms with running times ranging between $O(n^5)$ and $O(n^9)$. We conjecture that similar results can be obtained for polygonal regions using techniques similar to those presented in this thesis. The obvious question of the computation of
a largest minimum diameter spanning tree is open. We note that a straightforward generalization of the algorithms in the maximization variant is not correct, since there may exist cases for which Fact 6.10 does not hold (see Figure 6.12 for such an example).

**List of Contributions**

The results presented in this thesis were obtained throughout the four years of my doctoral studies. Even though the name on the front page is mine, I was not the only person involved in their development. In the following, I try to allocate the merit of each contribution with its respective author.

**Chapter 1.** A survey on routing under uncertainty was originally written as part of a report in the scope of the project eCOMPASS [1]. Most of the research on the related work and relevant literature was done by me. Yann Disser and Matúš Mihalák operated as reviewers and editors of the write-up.

**Chapter 2.** The ideas for the enumeration and the bi-directional algorithms BiTdMARTINS are joint work with Matúš Mihalák. The C++ implementation of these algorithms as well as the experimental evaluation on the road network of the Berlin and Brandenburg area were carried out by me. The results on the algorithm BiTdMARTINS of Section 2.3 are soon to be published in [101].

**Chapter 3.** The study of the Markov chain Monte Carlo technique for sampling simple paths in planar graph is joint work with Paolo Penna and has been published in [102]. A first rough draft of the proofs of Theorems 3.5 and 3.12 was done by me and later neatly polished and much improved by Paolo. The bottleneck ratios of Theorem 3.7 and Corollary 3.8 are by me and Paolo, respectively. The nice pictures of this chapter were drawn by Paolo.

**Chapter 5.** The results in this chapter are joint work Yann Disser, Matúš Mihalák and Peter Widmayer. A preliminary NP-hardness proof of Theorem 5.4 was designed by Yann Disser and later strengthened
as a hardness of approximation result by me and Matúš. The results presented in Section 5.1 can be found in [40] and those of Section 5.2 in [39].

**Chapter 6.** A previous NP-hardness proof of Theorem 6.1 for a more restricted case came from the mind of Yann Disser. Me and Matúš later generalized the result and strengthened it to the non-existence of an FPTAS. We published this result in [39]. The proofs of Theorems 6.5 and 6.6 belong to the encyclopedic knowledge on Voronoi diagrams of Chih-Hung Liu, as well as the idea of applying these structures to the computation of minimum placements for minimum diameter spanning trees. The same author is also the originator of the ideas behind Lemmas 6.11, 6.14, 6.15 and 6.21, and the implementation of the elementary operations of Section 6.3. The results of Section 6.2 are soon to be published in [89].


