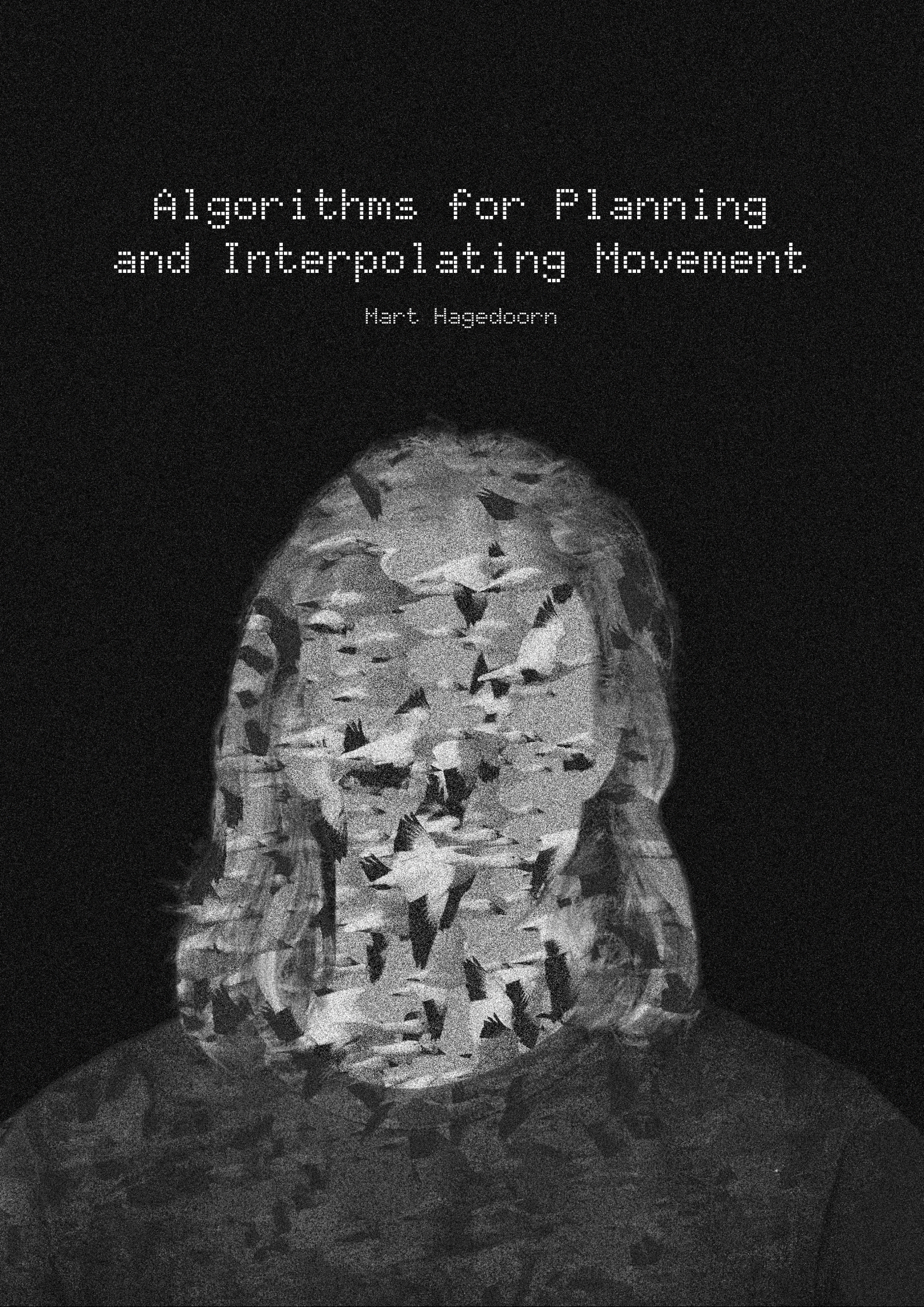
# Algorithms for Planning and Interpolating Movement

Mart Hagedoorn



# Algorithms for Planning and Interpolating Movement

## Dissertation

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

der Technischen Universität Dortmund
an der Fakultät für Informatik

von
Mart Hagedoorn

Dortmund
2026

| | |
|---|---|
| Tag der mündlichen Prüfung: | 31.07.2026 |
| Dekan: | Prof. Dr. Jens Teubner |
| Gutachter: | Prof. Dr. Kevin Buchin |
| | Prof. Dr. Valentin Polishchuk |

# Abstract

This thesis studies algorithmic problems arising in route planning and movement analysis in discrete and geometric environments. The common goal is to develop efficient methods for computing, approximating, or analyzing paths and walks under structural, temporal, and geometric constraints. The results range from complexity and approximation algorithms for route optimization problems to practical tools for tour generation, stochastic trajectory reconstruction, and geometric shortest-path queries.

A central part of this thesis is devoted to variants of the Orienteering Problem, where the task is to find a route of limited length that maximizes collected profit. First, we extend the Orienteering Problem to include time windows and investigate tractable and intractable cases on restricted graph classes such as directed and undirected paths, cycles, and trees. We give an efficient algorithm for directed paths, approximation and polynomial-time approximation schemes for directed cycles, and further results for edge-based time windows. For the classical Orienteering Problem without time windows, we show hardness on trees with unit profit and unit edge cost, but also provide a polynomial-time approximation scheme for bounded-treewidth instances. These results identify the boundary between polynomial-time solvability and hardness, while showing how structural graph restrictions can still be exploited algorithmically. Complementing this, this thesis presents a $(4 + \varepsilon)$-approximation algorithm for the Edge Orienteering Problem, where profits are associated with edges and traversal costs may be incurred multiple times.

Beyond these theoretical contributions, we also address practical route generation. We present a flexible framework for the Touring Problem, in which the objective is to compute round trips of a prescribed length according to user preferences. We use the framework to do an experimental study of the Touring Problem.

This thesis further develops an algorithmic framework for reconstructing movement from sparse trajectory data using conditioned random walks. Starting from discretized transition models, we give dynamic-programming methods for computing transition densities, visit probabilities, and utilization distributions between time-stamped observations. The framework supports not only Brownian motion, but also biased and correlated random walks, mixtures of movement models, and state- or habitat-dependent movement models. These methods enable the efficient generation of individual conditioned walks as well as the computation of occupancy and visitation probabilities. Furthermore, we demonstrate the methods by performing an ecological case study on Eastern Bettongs showing how we can support habitat-use and behavioral-state analysis from sparse GPS trajectories.

Finally, we study minimum-link paths in polygonal domains with holes. We show how to preprocess such a domain so that the link distance between two query points can be reported efficiently. Using the resulting data structures, the link diameter, center, and radius of the domain can be computed in polynomial time. In addition, this thesis gives

a simpler algorithm for computing the link diameter that does not rely on the query structures. These results resolve open questions on 2-point link distance queries and on computing the link diameter, radius, and center in polygonal domains with holes.

Taken together, this thesis advances the algorithmic foundations of path and walk computation across graph optimization, stochastic movement modeling, and computational geometry, combining structural complexity results with efficient algorithms and practically usable systems.

# Acknowledgements

A bit more than four years have passed and I have reached the end of my PhD project. These have been years of ups and downs, but have, above all, been incredibly informative. However, I could not have done this by myself and am grateful for all the people that got me here. I will try to thank as many people as I can, but this list is by no means exhaustive.

First I would like to thank my supervisor Kevin, who had the most essential role in my path to completing my PhD. I thank you for not just helping with my research, but also for giving me the freedom to pursue my own ideas and keeping me engaged with the different projects I have been involved in over the years.

I am also grateful for the colleagues of our research group. First, Carolin for your guidance and sharing of knowledge on how to navigate the academic world. Of course, I would also like to thank Antonia for sharing this PhD journey with me. I cannot say enough how nice it has been to have someone going through the same experiences at the same time. Next, I would like to thank Guangping and Torben for helping create such a pleasant work environment. Finally, I would like to thank Betty for all her support and for giving me such a good start on my German learning journey.

Also many thanks to Valentin and Emiel for hosting me on my research visits. In these visits I learned a lot and was very happy for your time.

Nächstens möchte ich gerne Henning danken, du warst mein erster Freund in Dortmund der nicht mit der Universität verbunden war, und hast an meine Seite gestanden während das ganze Prozess. Auch möchte ich gerne Linus und Tobi danken für all die gemeinsame Kletterzeit und Reisen, ihr habt mich immer aufgefangen, wenn ich abgestürzt bin. Neben das Klettern ist auch das Tanzen ein riesiger Teil meines Leben geworden und möchte spezifiezisch Olivia danken für all die Stunden die wir zusammen getanzt und gechillt haben. Und weiter möchte ich auch noch gerne Hilde, Maria, Lena, Max, Stella, Felix, Julia, Martin und eigentlich noch viele weitere Personen danken für all die schönen Tanz-, Kletter-, Brettspielzeiten (ihr wisst wo ihr rein geordnet sein sollte).

Vervolgens wil ik ook graag mama en papa bedanken voor jullie constante steun en ongelimiteerde liefde. En natuurlijk ook Lotte en Joost voor dezelfde dingen als m'n ouders, maar ook voor het support dat ik gevonden heb doordat we allebei in dezelfde fase zitten in ons leven. En natuurlijk kan ik niet Max en Laura vergeten, ondanks de afstand hebben we elkaar altijd veel gezien en heb ik veel uit onze gezamenlijke liefde voor bordspellen en boeken gehaald.

And last but not least, I would like to thank you, Kate, for standing by my side during the final part of this journey. I am very happy we happened to be in Santander at the same time, and that you also thought it was a good idea to host an almost complete stranger for a while. The last year has meant a lot to me, and you have been my greatest support during one of the most stressful periods of my life.

# Contents

# Chapter 1
# Introduction

In our physical world, almost nothing is purely stationary. If we follow Heraclitus' famous dictum *πάντα ῥεῖ* ("everything flows"), then change is not an exception but the default. It is therefore unsurprising that movement has been a topic of thought for a very long time. Whilst philosophical reasoning about movement appears in early Western philosophy, humans have likely engaged with questions of movement far earlier, for instance through animal tracking and hunting practices, which require inferring unseen trajectories [75]. Across these different contexts, we see that movement is fundamental to humans and to life around us, and understanding movement better is one way of better understanding the world we live in.

Although people have reasoned about these topics for a very long time, questions about motion have become increasingly prominent. Where prehistoric humans had very practical concerns about movement and the Ancient Greeks asked largely philosophical questions, today's researcher is confronted by an abundance of *data* about movement. Large parts of the world's population carry smartphones that record locations at high resolution. In ecology, GPS trackers are attached to animals to study migration, habitat use, and the impact of human activity. Even when GPS is unavailable, camera traps, photo timestamps, and other metadata can provide observations from which we still try to study an individual's trajectory.

Collecting movement data is only useful, however, if we can analyze it. Real-world datasets are often incomplete: devices run out of battery, sensors fail, individuals must be protected for privacy reasons, or observations are irregular. This creates a central challenge: how can we infer, compare, and analyze movement when we only observe fragments? A common first step is *interpolation*, i.e., estimating what happened between two recorded points. The simplest approach connects endpoints by a straight line segment, but this is often unrealistic: obstacles block direct travel, agents avoid or are attracted to certain regions, and non-linear routes can be more efficient. Thus, gaps between observations force a balance between three competing goals: trajectories should be optimal with respect to an objective, plausible given partial evidence, and simple enough to be interpretable and executable.

Looking forward, reasoning about movement is not limited to reconstructing past trajectories; it also includes planning future ones. Path planning is a central topic in computer science, ranging from shortest paths to variants that optimize multiple criteria. The classical starting point for path planning is the shortest path problem: given a space equipped with a cost function (such as Euclidean distance or travel time), we want to find the cheapest route between a start and an end point. Even in this basic formulation, the notion of "cost" is application-dependent. In road networks it may represent travel time under congestion; in robotics it may incorporate clearance from obstacles or risk; and in

hiking or cycling it may penalize steep slopes or difficult terrain. In many settings, costs are therefore not purely geometric but encode preferences and constraints that reflect how an agent actually moves.

Instead of minimizing cost, there are problems that aim to maximize some reward instead. In many settings, such as tourism, recreation, and ecological surveying, the goal is not to reach the destination as quickly as possible, but to collect as much value within limited resources such as time, distance, or battery. For example, someone might want to visit as many points of interest during a short city trip or animals may follow routes that balance energy-efficiency and access to food or shelter. These settings naturally lead to *budgeted routing problems* such as the Orienteering Problem. Moreover, rewarding movement is not just about where one goes, but also how one gets there; some streets are scenic whilst others are safe. These settings motivate the Edge Orienteering Problem, where rewards are assigned to edges instead of vertices.

In this thesis, I look at movement in different settings through three related algorithmic lenses: optimality, plausibility, and simplicity. By optimality, I ask how to choose trajectories which best satisfy an objective under specific constraints and limited resources. Plausibility asks how to best reason about trajectories when parts of the trajectories are unobserved. Finally, simplicity asks about keeping the geometric complexity of a trajectory small without ignoring the surrounding environment. Throughout the thesis, the goal is to understand how, in different settings, the underlying space can be structured and used when trying to predict or compute movement. Concretely, this thesis contributes algorithmic results in three settings: budgeted reward-maximizing routing (Orienteering/Touring), probabilistic interpolation of sparse trajectories (random walks) and turn-minimizing motion planning in polygonal domains (Minimum-Link Paths).

**Orienteering and Touring.** The first setting we consider is the *Orienteering Problem*, where the goal is to select a route under a travel budget that maximizes a collectible reward placed on vertices of a graph. The Orienteering Problem originates from the sport of orienteering: Competitors navigate between control points marked on a map, aiming to visit as many points as possible within a given time limit. However, the problem is often applied in the context of tourism [73, 91], logistics [41, 87], and journey planning [MH5, 59].

In many of these cases the class of graphs to be considered can often be restricted. For instance, when planning a tour from A to B, one can assume that the tour stays close or even largely makes use of the main route between A and B, while taking small detours to visit interesting sites close to this main route. Thus, we can limit the problem to a path-like or tree-like subgraph of the travel network. This scenario could be well modeled by graphs with bounded treewidth. Moreover, in many real-world situations points of interest are only interesting at certain times. For example, a restaurant is only interesting during opening hours, and a park may be more interesting during the day than at night. This motivates the Orienteering Problem with time windows, where each reward can only be collected within certain time intervals.

However, when the journey rather than the destinations becomes the goal, one might be more interested in the route traveled. For example, someone doing a hiking trip would likely prefer to traverse quiet roads through a forest instead of city streets buzzing with cars. Hence, in these cases it makes more sense to look at the *Edge Orienteering Problem*, where the rewards are collected by traversing edges instead of visiting vertices. Further, end users might be more interested in more than distance and road types alone. Users often prefer a combination of quiet roads, scenic segments, and routes that encourage exploration. To capture such preferences, we extend the classical Edge Orienteering

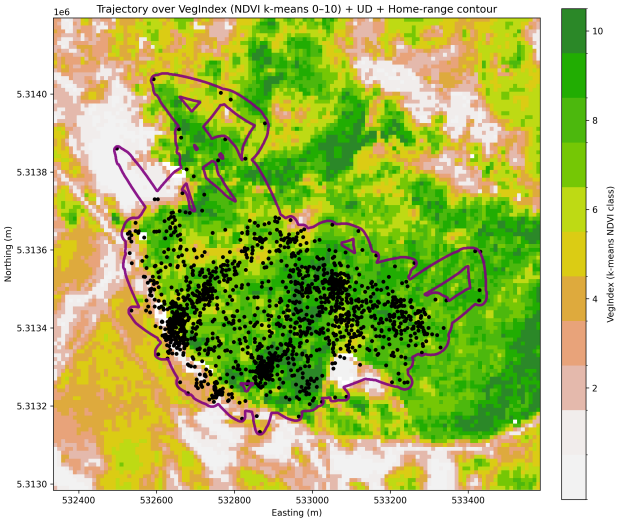



Figure 1.1: Home range (purple boundary) and location measurements (black) of an Eastern Bettong.

Problem by incorporating global quality measures that describe the overall structure of a route. We refer to this extension as the *Touring Problem*.

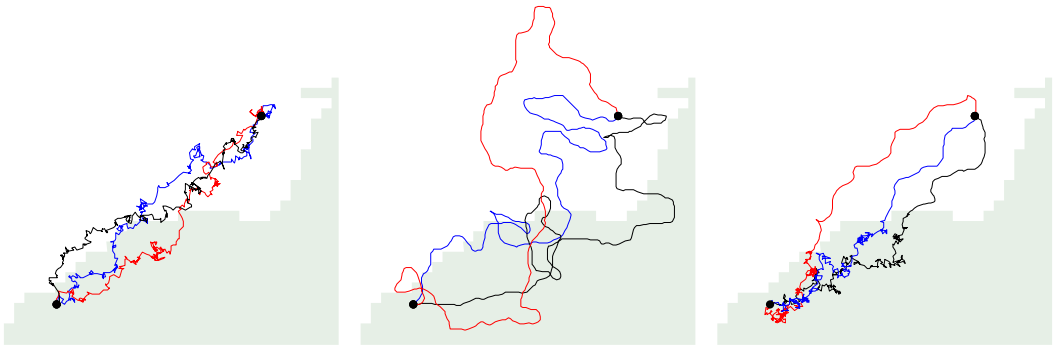
**Conditioned Random Walks.** The second movement model considered in this thesis is less goal-driven and aims at understanding behavioral aspects by analyzing trajectories. However, when recording human or animal movement, there are many situations in which the data obtained are sparse, meaning that the recorded location points can be spaced far apart with a significant temporal gap between two measurements. This can be caused by privacy concerns or practical problems such as the battery life[1] or signal reception of GPS trackers.

Yet, to better understand the moving agent the unobserved movement between measurements is often of primary interest. For instance, one may want to estimate whether a tracked entity has visited a certain area, a use case that is particularly interesting to validate exposure to diseases in certain regions. Another reason to look at the unobserved movement is to use this to estimate the so-called *home range* of an animal, which is the area in which that animal lives [19] and that it does not normally leave (See Figure 1.1).

This raises the question of how to compute, or estimate, the movement between several discrete points in space-time. As mentioned before, the first approach that may come to mind is to connect each point with its successor by a straight line. Such a linear movement model is probably the most commonly used model [65]. Although it provides a reasonable approximation of the movement when locations are measured at a sufficiently high frequency, this is usually not how humans or animals move [89, pp. 36–37] and may lead to incorrect analysis results [18].

The solution to this problem as considered in this thesis is to use *random walks* to model

[1] Without affecting their natural behavior, animals can carry at most 5 % of their body weight [83]. As a result, often very small batteries have to be used in GPS trackers.



(a) Brownian motion (b) Correlated motion (c) Field type aware motion

Figure 1.2: Examples of random walks with different movement models.

movement between two data points. The concept of random walks was first introduced in 1905 by Pearson [68] and refers to a walk that consists of multiple steps, with the direction and length of each step being chosen at random. However, these walks raise an additional challenge: typical random-walk models are designed to simulate unconstrained movement forward from a given starting location. They are not inherently suited to interpolate between two fixed measurements in space-time. In particular, a random walk that starts at the location and time of the first measurement will generally not reach the location of the second measurement after some prescribed time. Existing interpolation approaches therefore either rely on simple grid walks or employ heuristic corrections that steer the walk towards the second measurement.

In more advanced models, the probabilities for moves in certain directions can vary depending on the direction or other parameters (see Figure 1.2 for example paths). Random walks allow for the simulation of the movement behavior of an observed entity, providing simple models, which can easily incorporate knowledge about the moving entity (e.g. navigational capacity, behavior) and the environment (e.g., wind, land cover). Random walks require relatively little data to be learned, and are transferable across similar datasets. A large variety of random walk models are used to simulate movement [25], from simple Brownian motion to biased and correlated random walks and more complex models. In particular, state-space models [67] play a prominent role in modeling movement, as they provide means to incorporate external factors and behavioral states into the model.

**Minimum Link Paths.** Finally, we consider shortest paths that are constrained to consist of straight-line segments, where the objective is not to minimize the Euclidean length of the path but rather the number of segments (equivalently, the number of turns). A path with the minimum number of line segments is called a *minimum-link* path. We study minimum-link paths in polygonal domains, where they naturally arise in the motion planning of robots.

When planning trajectories of mobile robots, turns are often expensive operations. Executing a turn can require deceleration and recalibration of the sensors. Therefore, reducing the number of turns in a path can improve energy efficiency and reduce cumulative localization and sensing errors.

Minimum-link paths also appear beyond robotics and can be useful in ecological settings. For example, these paths are used when computing sweeping strategies with a minimum number of chain guards [82], i.e. a polyline ”curtain” that scans an environ-

ment and guarantees a moving agent will be found. This provides a useful abstraction for ecological surveying scenarios, where a team of agents sweeps a region to detect or contain moving entities while minimizing maneuvering complexity.

## 1.1 Contributions

In Chapter 2, we study the Orienteering Problem on restricted graph classes. While the Orienteering Problem with time windows is NP-hard even on undirected paths and cycles, and remains so even if all profits must be collected, we show that for directed paths it can be solved in $\mathcal{O}(m \log m)$ time (where $m$ is the total number of time windows), even if each profit can be collected in one of several time windows. The same case is shown to be NP-hard for directed cycles.

Particularly interesting is the Orienteering Problem on a directed cycle with one time window per profit. We give an efficient algorithm for the case where all time windows are shorter than the length of the cycle, resulting in a 2-approximation for the general setting. Based on the algorithm for directed paths, we further develop a polynomial-time approximation scheme for this problem. For the case where all profits must be collected, we present an $\mathcal{O}(n^4)$-time algorithm to decide whether there exists a feasible tour that visits all vertices within their time windows.

In the variant without time windows, we show that on trees and thus on graphs with bounded treewidth the Orienteering Problem remains NP-hard. We present, however, an FPT algorithm to solve the Orienteering Problem with unit profits that we then use to obtain a $(1 + \varepsilon)$-approximation algorithm on graphs with arbitrary profits and bounded treewidth, which improves current results on general graphs.

This chapter is based on [MH6]
K. Buchin, M. Hagedoorn, G. Li, and C. Rehs. “Orienteering (with Time Windows) on Restricted Graph Classes”. *SOFSEM 2025* 151–165, 2025.

In Chapter 3, we alter the Orienteering Problem by adding collectible reward to the edges instead of the vertices and additional rewards on the general structure of the tour, resulting in the Edge Orienteering Problem and Touring Problem, respectively. We present a $(4 + \varepsilon)$-approximation algorithm for the Edge Orienteering Problem and perform an experimental study on the Touring Problem for real-world instances.

This chapter is based on [MH3]
K. Buchin and M. Hagedoorn. “Improved Approximation Algorithms for the Edge Orienteering Problem”. *IPL 2026*. 195, 2026.
and [MH5]
K. Buchin, M. Hagedoorn, and G. Li. “Tour4Me: a framework for customized tour planning algorithms”. *SIGSPATIAL 2022*. 1–4, 2022.

In Chapter 4, we present a grid-based approach for efficient movement interpolation using conditioned random walks. The model is able to compute conditioned Brownian walks, conditioned correlated random walks, and more generally conditioned random walks where the underlying movement models change based on spatial and/or temporal data. Using the grid-based approach, it is possible to generate random walks conditioned to start and end destinations, calculate utilization distributions, and visit probabilities of certain target areas.

Furthermore, we demonstrate how our method can be used to perform an ecological study to analyze the behavioral patterns of a small terrestrial mammal. From this study

we conclude that the Eastern Bettong primarily prefers high canopy cover whilst foraging.

This chapter is based on [MH4]
K. Buchin, M. Hagedoorn, and A. Korn. "Discretized Random Walk Models for Efficient Movement Interpolation". *SIGSPATIAL 2024*. 102–112, 2024.

In Chapter 5, we show how to preprocess a polygonal domain with holes so that the link distance (the number of links in a minimum-link path) between two query points in the domain can be reported efficiently. Using our data structures, the link diameter of the domain (i.e. the maximum number of links that may be required in a minimum-link path between two points in the domain) as well as the link center and radius of the domain (i.e. the point minimizing the maximum link distance to the furthest point in the domain and this maximum link distance) can be found in polynomial time. We also give a simpler algorithm for finding the link diameter, that does not use the link distance query structures. Answering 2-point link distance queries and computing the link diameter/radius/center in polygonal domains have been open questions since these problems were studied for simple polygons in the 1990s.

This chapter is based on [MH7]
M. Hagedoorn and V. Polishchuk. "Link Diameter, Radius and 2-Point Link Distance Queries in Polygonal Domains". *WADS 2025*. 34:1-34:15, 2025.

For all papers discussed in this section, I contributed to the conceptualization, development of technical details, writing, and, where applicable, the implementation and experimental evaluation. Furthermore, I have contributed to two more peer-reviewed publications related to the broader research topic. These works are not part of the thesis and are therefore not discussed in the following chapters.

A. Atak, K. Buchin, M. Hagedoorn, J. Heinrichs, K. Hogreve, G. Li, and P. Pawelczyk. "Computing Maximum Polygonal Packings in Convex Polygons Using Best-Fit, Genetic Algorithms and ILPs (CG Challenge)". *SoCG 2024*. 83:1–83:9, 2024. [MH1]
and K. Buchin, J. Conradi, L. Deryckere, M. Hagedoorn, C. Rehs. "Tight Lower Bound for Approximating (k, ℓ)-Center Clustering under the Fréchet Distance". *EGC 2025*. [MH2]

# Chapter 2
# The Orienteering Problem

In this chapter we study the *Orienteering Problem* and several of its variants. The Orienteering Problem is a well-researched problem with a wide range of applications. The Orienteering Problem originates from the sport of orienteering: competitors navigate between control points marked on a map, aiming to visit as many points as possible within a given time limit. With the addition of different profits for each control point, this is a frequently applied routing problem, for instance in logistics [41, 87], tourism [73, 91], and journey planning [MH5, 59]. In such applications, the class of graphs to be considered can often be restricted. For instance, when planning a tour from A to B, one can assume that a nice tour stays close or even largely makes use of the main route between A and B, whilst taking small detours to visit interesting sites close to this main route. Thus, we can limit the problem to a path-like or tree-like subgraph of the travel network. This scenario is well modeled by graphs with bounded treewidth.

Formally, in the Orienteering Problem we are given a graph $G$, a start vertex $s \in V(G)$, a budget $B \in \mathbb{R}_{>0}$, an edge-cost function $c : E(G) \to \mathbb{R}_{\geq 0}$, and a profit function $\pi : V(G) \to \mathbb{R}_{\geq 0}$. A feasible solution is a walk $P = (e_1, \ldots, e_\ell)$ starting at $s$ such that

$$\sum_{i=1}^{\ell} c(e_i) \leq B.$$

Let $V(P)$ denote the set of vertices visited by $P$ (including $s$). The profit of $P$ is

$$\pi(P) = \sum_{v \in V(P)} \pi(v),$$

i.e., each vertex profit is collected at most once. The goal is to find a feasible walk maximizing $\pi(P)$.

The Orienteering Problem has been a popular topic of research over the last decades (see [90] for a survey). Considerable effort has been devoted to designing exact (exponential-time) algorithms [31, 32, 57, 69] and practical heuristics [4, 74]. Since –as a generalization of the Traveling Salesperson Problem– the Orienteering Problem is NP-hard [41], theoretical work on the Orienteering Problem has mostly focused on approximation algorithms. The current best approximation algorithm is a $(2 + \varepsilon)$-approximation by Chekuri, Chekuri, and Pál [21]. However, the best lower bound on the approximation ratio is 1481/1480, given by Blum, Chawla, Karger, Lane, Meyerson, and Minkoff [13]. Furthermore, when the points of interest lie in a fixed-dimensional Euclidean space and the underlying graph is the complete Euclidean graph, a $(1 + \varepsilon)$-approximation algorithm has been proposed by Chen and Har-Peled [22].

A natural extension of the Orienteering Problem is to consider *time windows*, i.e. time intervals. In applications, points of interest often need to be reached in specific time windows due to the availability of certain services, hours of operation, or deadlines.

In the Orienteering Problem with time windows, additionally each profit is associated with one or more time windows, and these profits can only be collected if the walk visits the corresponding vertex during one of these windows. We distinguish between the version in which every vertex has one single time window (OP-1TW) and the version with multiple time windows per vertex (OP-MTW).

A restricted version of the Orienteering Problem with time windows, which is closely related to the Traveling Salesperson Problem with time windows, asks if there is a walk $P$ which collects the profit of every vertex. We refer to these problems as the Covering Orienteering Problem with single/multiple time windows (COP-1TW / COP-MTW) and show several hardness results even for these restricted problems.

Problems closely related to the Orienteering Problem with time windows have been studied before. The most frequently studied related problem is the Traveling Salesperson Problem with time windows [1, 9, 12]. Since NP-hardness for the Traveling Salesperson Problem implies NP-hardness for these versions with time windows, research has been primarily focused on heuristics (see [43] for a survey), approximation approaches [9], or algorithms on restricted graph classes [88]. Most notably, the work done by Tsitsiklis [88] is strongly related to the COP-1TW, since, in some variations discussed by Tsitsiklis, a walk is allowed to visit the same vertex multiple times. Moreover, Garg, Khanna, and Kumar [35] introduced the multiple time window model for Orienteering, proving it is APX-hard on trees.

While approximation algorithms for the Orienteering Problem with time windows have been considered before, little is known about this problem on restricted graph classes. This is particularly surprising since the problem with time windows is interesting from both an application and an algorithmic point of view even on very restricted graph classes: here, an optimal route might pass an important vertex at a "wrong" time and have to come back later.

In this chapter, we investigate the Orienteering Problem with time windows. For an overview of the presented results, see Table 2.1. Even on undirected paths, COP-1TW and OP-1TW are NP-hard, which follows from a similar problem considered in [88]. Furthermore, we show that COP-MTW and thus OP-MTW are NP-hard even on directed cycles. Perhaps surprisingly, this is not true for one single time window per vertex: We give a polynomial-time algorithm for the COP-1TW. Moreover, we present a dynamic program for OP-MTW on directed paths. While the question of NP-hardness remains open for OP-1TW on directed cycles, we give an $\mathcal{O}(n^2 \log n)$-time algorithm for short time windows (which admits an FPT algorithm with respect to the number of long intervals), a simple 2-approximation, and a polynomial-time approximation scheme.

Another natural setting in applications, which is closely related to having time windows for the profits, is that edges cannot be used at all times, for example due to construction sites, road illumination or seasonal limitations. Formally, we obtain the Orienteering Problem in a setting of dynamic graphs (OP-D for short) by adding one or more intervals to each edge in which that edge is traversable. Surprisingly, while the setting seems very similar to the Orienteering Problem with time windows, we show that it is solvable in quadratic time on an undirected path. However, the Orienteering Problem on dynamic graphs remains NP-hard on trees even with unit profit and unit cost.

We further consider the Orienteering Problem without time windows. We show that although this problem is NP-hard even on trees (and thus also on graphs with bounded

treewidth), we give a $(1+\varepsilon)$-approximation algorithm on graphs with bounded treewidth. This stands in contrast to the Orienteering Problem on general graphs that does not admit a PTAS unless P=NP.

## 2.1 The Orienteering Problem with Time Windows

In this section, we study the Orienteering Problem with time windows. We distinguish between the version of the problem using a single time window per profit (OP-1TW) and using multiple time windows (OP-MTW). An instance of OP-1TW consists of an instance of the Orienteering Problem together with an interval $[r_i, d_i]$ as time window for each of the vertices $v_i \in V(G)$. A walk $W$ through $G$ is now defined by a sequence of tuples $(v, t)$, where $v$ is a vertex and $t \in \mathbb{R}_{\geq 0}$ is a *time* such that if $(v, t)$ is followed by $(v', t')$ in $W$, then $t' \geq t + c(\{v, v'\})$. When $t' > t + c(\{v, v'\})$, the walk waits at $v$ from time $t$ to $t' - c(\{v, v'\})$, then traverses $(v, v')$ and arrives at time $t'$. The profit of a vertex $v_i$ can only be collected once by $W$ and only if $W$ passes $v_i$ at time $t$ with $r_i \leq t \leq d_i$.

The OP-MTW generalizes the OP-1TW by replacing the time interval per vertex by a set of disjoint time intervals $\mathcal{I}_{v_i} = \{[r_{i,1}, d_{i,1}], \cdots, [r_{i,m_i}, d_{i,m_i}]\}$, where $m_i$ is the number of time windows of $v_i$. The profit of vertex $v_i$ can then be collected at any time $t$ for which there is an interval $[r_{i,j}, d_{i,j}] \in \mathcal{I}_{v_i}$, such that $r_{i,j} \leq t \leq d_{i,j}$. Furthermore, let $d_{max}$ be the latest deadline of all time windows. We can cap every deadline above $B$ to $B$ (and ignore time $> B$); w.l.o.g. assume that budget $B = d_{max}$.

As we will see in Section 2.3, the Orienteering Problem in general is NP-hard even on trees, and thus remains NP-hard in the version with time windows. However, in the setting of Orienteering Problem with time windows, the problem is NP-hard even for undirected paths, which follows directly from the following. The line-TSP problem as described and proven NP-complete by Tsitsiklis [88] is closely related to the OP-1TW on undirected paths even with unit edge cost and unit vertex profit. In line-TSP, each job $j \in J$ has an integer position $x_j$ and time window $[r_j, d_j]$ and travel time between two jobs $j, j' \in J$ is defined as $|x_j - x_{j'}|$. The goal in line-TSP is to find a walk which collects the profit of all the jobs in $J$. The difference to the OP is that in line-TSP two jobs can have the same position, which is not possible when each vertex has exactly one associated profit and all edges have unit costs. However, it is possible to reduce line-TSP to the OP-1TW on an undirected path with unit edge cost and unit vertex profit by a straightforward reduction.

**Remark 1.** *The OP-1TW, and hence OP-MTW, is weakly NP-hard on undirected paths even with unit edge costs and unit vertex profits.*

| | Single time window | Multiple time windows |
|---|---|---|
| Directed path | $\mathcal{O}(n \log n)$, Prop. 2 | $\mathcal{O}(m \log m)$, Prop. 2 |
| Directed cycle | COP-1TW: $\mathcal{O}(n^4)$, Thm. 4<br>OP-1TW: PTAS, Thm. 11<br>FPT, Cor. 6 | NP-hard, Thm. 12 |
| Undirected path | NP-hard [88] | NP-hard [88] |

Table 2.1: Results for the Orienteering Problem with a single or multiple time windows per vertex, where $m$ is the total number of time windows.

*Proof.* We reduce from the feasibility problem for *line-TSP* with zero processing times, as defined by Tsitsiklis [88]. Consider a line-TSP instance consisting of a set $J$ of $n$ jobs. Each job $j \in J$ has an integer position $x_j$ and an integer time window $[r_j, d_j]$. The server starts at position $x^*$ at time zero, and the travel time between two positions $x$ and $x'$ is $|x - x'|$.

Set

$$M := 2n^2 + 1, \qquad C := n^2, \qquad d_{\max} := \max_{j \in J} d_j,$$

and let

$$B := M d_{\max} + C.$$

For every job $j \in J$, introduce a corresponding vertex $v_j$. In addition, introduce a starting vertex $s$ corresponding to the initial position $x^*$. Associate with these vertices the positions

$$\xi(v_j) := x_j \qquad \text{and} \qquad \xi(s) := x^*.$$

Sort the vertices in $\{s\} \cup \{v_j : j \in J\}$ by nondecreasing value of $\xi$, breaking ties arbitrarily.

We construct an undirected path $G$ by connecting every two consecutive vertices $u$ and $v$ in this order as follows. If $\xi(u) = \xi(v)$, we add the edge $(u, v)$. Otherwise, we connect $u$ and $v$ by a path containing

$$M\big(\xi(v) - \xi(u)\big) - 1$$

intermediate vertices of unit profit. Thus, the distance between two consecutive special vertices at distinct positions is

$$M\big(\xi(v) - \xi(u)\big).$$

Vertex $s$ has unit profit and time window $[0, 0]$, meaning this profit is always collected at the start of the walk. Furthermore, every job vertex $v_j$ has unit profit and time window

$$[M r_j, M d_j + C].$$

The intermediate vertices do not represent jobs, each intermediate vertex can have unit profit and be assigned the time window

$$[0, 0],$$

so that its profit cannot be collected by any walk starting at vertex $s$.

For any two job vertices $v_i, v_j$, and also when one of them is replaced by the starting vertex $s$, the construction satisfies

$$M|x_i - x_j| \le d_G(v_i, v_j) \le M|x_i - x_j| + n, \tag{2.1}$$

where $d_G(v_i, v_j)$ is the distance between $v_i$ and $v_j$ in $G$.

Suppose first that the line-TSP instance has a feasible solution visiting the jobs in the order $j_1, \dots, j_n$ at integer times $t_1, \dots, t_n$. Set

$$T_k := M t_k + kn.$$

For every $k$, feasibility of the original solution and (2.1) imply

$$T_k - T_{k-1} \ge M|x_{j_k} - x_{j_{k-1}}| + n \ge d_G(v_{j_{k-1}}, v_{j_k}),$$

where $v_{j_0} := s$ and $T_0 := 0$. Hence the corresponding vertices can be visited at times $T_k$.

Moreover,

$$Mr_{j_k} \le T_k \le Md_{j_k} + n^2,$$

so every job is collected within its new time window. Thus, all $n$ job profits (and the start vertex profit) can be collected.

Conversely, suppose that a walk collects all $n$ job profits (and the start vertex profit), in the order $j_1, \dots, j_n$, at times $T_1, \dots, T_n$. Define

$$t_k := \left\lfloor \frac{T_k}{M} \right\rfloor.$$

By the lower bound in (2.1), we have

$$T_k - T_{k-1} \ge M|x_{j_k} - x_{j_{k-1}}|,$$

and therefore

$$t_k - t_{k-1} \ge |x_{j_k} - x_{j_{k-1}}|.$$

Furthermore, since

$$Mr_{j_k} \le T_k \le Md_{j_k} + n^2 < M(d_{j_k} + 1),$$

we obtain

$$r_{j_k} \le t_k \le d_{j_k}.$$

Consequently, the jobs can be visited in the same order within their original time windows, giving a feasible line-TSP solution. Hence, the line-TSP instance is feasible if and only if OPT = $n + 1$. □

### 2.1.1 Directed Path

While on an undirected path the problem is NP-hard, the Orienteering Problem with time windows is solvable in polynomial time if the input graph is a directed path. This holds even for multiple time windows:

**Proposition 2.** *Given an instance of the OP-MTW with a directed path $G$, arbitrary profits, and arbitrary edge costs, the OP-MTW can be solved in $\mathcal{O}(m \log m)$ time, where $m$ is the total number of time windows.*

*Proof.* Assume w.l.o.g. that every vertex has at least one time window, i.e. $m \ge |V(G)|$. If a vertex $v \ne s$ has no time window, then its profit cannot be collected. Such a vertex can be contracted, replacing its two incident edges by one edge whose cost is the sum of their costs. Vertices preceding $s$ and an uncollectible terminal vertex can simply be discarded. Furthermore, let $V(G) = \{v_1, \cdots, v_n\}$, where $v_1 = s$, and $E(G) = \{(v_i, v_{i+1}) \mid 1 \le i < n\}$.

We specify a dynamic programming algorithm. For $i \in \{1, \dots, n\}$ and $t \in [0, d_{\max}]$, let

$$F_i(t) := \max\{\pi(W) \mid W \text{ is a feasible walk that ends at } v_i \text{ at time } t\}.$$

(If no such walk exists, set $F_i(t) = -\infty$.) Since waiting is allowed, $F_i(t)$ is nondecreasing in $t$ for every $i$. Let $D_i := \sum_{k=2}^{i} c(v_{k-1}, v_k)$ be the travel time from $v_1$ to $v_i$ along the path without waiting (with $D_1 := 0$). It is convenient to reparametrize time by the *slack* $\tau := t - D_i$ and to define

$$G_i(\tau) := F_i(D_i + \tau) \qquad \text{for } \tau \in [0, d_{\max} - D_i].$$

We use the slack parameterization only to simplify notation; all feasibility and profits are defined in absolute time via $F_i$.

**Data structure and invariant.** We maintain $G_i$ as a nondecreasing step function over $\tau \in [0, d_{\max}]$, where the function only changes at start- and endpoints of shifted time windows. In order to calculate $G_i$, we maintain a balanced dynamic segment tree $T$, where the nodes of $T$ correspond to time intervals. Specifically, at iteration $i$, a leaf $\ell$ of $T$ corresponds to interval $[\ell_l, \ell_r)$. Each internal node $w \in T$ is associated with the interval $[\ell'_l, \ell''_r)$, where $\ell'$ is the left endpoint of the leftmost leaf and $\ell''$ is the right endpoint of the rightmost leaf of the subtree rooted at $w$. Furthermore, internal nodes are also augmented with some profit value $\pi_w$ representing a lazy increment applied to the entire interval. We define query $T(\tau)$ to return the sum of profit values stored in the nodes in the path going from the root of $T$, down to the leaf $\ell$ such that $\tau \in [\ell_l, \ell_r)$.

We store values on half-open leaf intervals, but time windows are closed. Thus we treat each window $[r, d]$ as $[r, d + \varepsilon)$, for an infinitesimal $\varepsilon > 0$, so that feasibility at $t = d$ is preserved.

For iteration $i$, let $H_i := d_{\max} - D_i$ and $p_i := \pi(v_i)$. Then, the loop invariant at the beginning of iteration $i$ is:

$$T(\tau) = G_{i-1}(\tau) \quad \text{for all } \tau \in [0, H_i],$$

and $T(\tau)$ is nondecreasing in $\tau$.

**Initialization ($i = 1$).** Let $r_{\min}$ be the minimum release time among the windows of $v_1$. We create two leaves $[0, r_{\min})$ and $[r_{\min}, d_{\max} + \varepsilon)$ with values 0 and $\pi(v_1)$. This gives us exactly $G_1$ and $T(\tau)$ is nondecreasing in $\tau$. Since the walk starts at $v_1$ at time 0, for every $t \geq r_{\min}$ it must be at $v_1$ at time $r_{\min}$ and thus collects $\pi(v_1)$. Furthermore, we intialize $\Pi_{\max} := T(d_{\max})$ to keep track of the maximum profit collected across all iterations.

**Update step.** Assume $i \geq 2$ and that the invariant holds for $G_{i-1}$. First we compute the union $U_i$ of shifted windows of $v_i$ in $\tau$-coordinates. That is,

$$U_i := \left( \bigcup_{[r,d] \in \mathcal{I}_{v_i}} [\, r - D_i,\ d - D_i \,] \right) \cap [0, H_i]$$

.

We process each disjoint interval $[a, b]$ of $U_i$ from right to left as follows (interpreting it as half-open $[a, b+\varepsilon)$ in the tree). Let $R$ be the left endpoint of the interval of $U_i$ immediately to its right, if it exists, and let $R := H_i + \varepsilon$ otherwise. We first split the leaves containing $a$, $b + \varepsilon$, and $R$, if necessary, so that these values are leaf boundaries.

Thereafter, before modifying this interval, let $z := T(b) + p_i$ and we first apply a range-add of $p_i$ only to $[a, b + \varepsilon)$. Then, starting with the leaf whose left endpoint is $b + \varepsilon$, we delete consecutive leaves contained in $[b + \varepsilon, R)$ for as long as their value is smaller than $z$, merging them into the preceding leaf of value $z$. We stop upon reaching $R$ or a leaf whose value is at least $z$.

Since $T$ is nondecreasing, this replaces $T(\tau)$ by $\max\{T(\tau), z\}$ for all $\tau \in [b + \varepsilon, R)$. After processing all intervals from right to left, the resulting function is nondecreasing and equals $G_i$. We maintain a variable $\Pi_{\max} := \max\{\Pi_{\max}, T(H_i)\}$ after every iteration and return $\Pi_{\max}$ at the end.

**Correctness.** By the DP derivation above, $G_i$ is obtained from $G_{i-1}$ by adding $\pi(v_i)$ on the shifted-window union $U_i$ and then taking the prefix maximum. Each processed interval $[a, b]$ performs exactly this transformation on the maintained step function, and processing all intervals of $U_i$ from right to left realizes the union. Therefore after iteration $i$ the tree represents $G_i(\tau) = F_i(D_i + \tau)$ for all $\tau \in [0, H_i]$, i.e., the optimal profit for ending at $v_i$ at every absolute time. The optimal OP-MTW profit over all endpoints and times is therefore

$$\max_{i\in[n]} \max_{t\in[0,d_{\max}]} F_i(t) = \max_{i\in[n]} \max_{\tau\in[0,d_{\max}-D_i]} G_i(\tau),$$

which we can obtain by querying the structure at $\tau = d_{\max} - D_i$.

**Runtime.** Across all vertices, we split leaves only at start- and endpoints of shifted time windows. There are $O(m)$ such start- and endpoints, hence $O(m)$ splits. Each split costs $O(\log m)$. In the domination loop, each deleted leaf is removed permanently; since the tree starts with $O(1)$ leaves and only $O(m)$ leaves are ever created by splitting, the total number of deletions over the entire algorithm is $O(m)$. Each deletion costs $O(\log m)$ to locate and remove. Finally, each range-add update is applied to $O(\log m)$ canonical subtrees, hence costs $O(\log m)$ per processed interval, and the total number of processed (merged) intervals over all vertices is $O(m)$. Thus the overall runtime is $O(m \log m)$. □

### 2.1.2 Directed Cycle

So far we have seen that on an undirected path, the Orienteering Problem with time windows is NP-hard, while on a directed path, it is solvable in polynomial time (regardless of the number of time windows). What seemingly makes the problem for directed paths easier is that we cannot revisit vertices.

A natural question now is what happens for directed cycles. In this graph class, a walk can revisit vertices. But in contrast to an undirected path, decisions are more limited because we can only walk in one direction (or wait at a vertex). We will see that this already makes the problem more complicated to solve than for directed paths. Therefore, we start with the Covering Orienteering Problem (COP) with time windows, which asks if there is a walk such that the profit of every vertex can be collected. Throughout this section, let $C = \sum_{e\in E(G)} c(e)$ denote the time needed to traverse the cycle once. Furthermore, for each $1 \le i, i' \le n$ let $d(v_i, v_{i'})$ be the distance of the unique directed path from $v_i$ to $v_{i'}$.

#### Single Time Windows

We start by considering the COP-1TW model on directed cycles, i.e. every vertex $v_i$ has exactly one time window $[r_i, d_i]$. Recall that a COP-1TW instance is a yes-instance if and only if there exists a walk where all profits are collected.

**Lemma 3.** *Given an instance $I$ of the OP-1TW on a directed cycle with $n$ vertices, there exists a reduced instance $I'$ whose latest deadline is at most $4nC$, such that $OPT = OPT'$, where $OPT$ and $OPT'$ are the optimal profits of $I$ and $I'$, respectively. Moreover, $I$ is a yes instance of COP-1TW if and only if $I'$ is a yes instance.*

*Proof.* Given an instance $I$ of the OP-1TW, we encode $I$ as a sequence of events as follows. Take the start event $t_0 = 0$, all release times and deadlines of instance $I$ and sort them in ascending order. Let $\{t_0, t_1, t_2, \cdots, t_{2n}\}$ be the resulting sequence, denoted as the *event*

*sequence* of $I$. We obtain the event sequence $\{t'_0, t'_1, t'_2, \cdots, t'_{2n}\}$ of the reduced instance $I'$ as follows: Start with the first event and set $t'_0 := t_0$ and then check each consecutive pair $t_i, t_{i+1}$ (these can be either release times or deadlines) for $i < 2n$ as follows. If $t_{i+1} - t_i \leq 2C$, then set $t'_{i+1} := t'_i + t_{i+1} - t_i$. Otherwise, if $t_{i+1} - t_i > 2C$, then we update $t'_{i+1} := t'_i + 2C$. Note that after applying this for each of the $2n$ consecutive pairs, the last event (the latest deadline) is at most $4nC$.

We now construct the reduced instance $I'$. The underlying directed cycle (and all edge costs) remain unchanged. For each vertex $v$ with time window $[r_v, d_v]$ in $I$, let $r_v = t_a$ and $d_v = t_b$ for the corresponding event times in the event sequence. We define the time window of $v$ in $I'$ as

$$[r'_v, d'_v] := [t'_a, t'_b].$$

Since the order of the events remains the same after the reduction, we get the following key observation: For each vertex $v_j$ and each event gap $[t_i, t_{i+1}]$, the time window of $v_j$ includes the interval $[t_i, t_{i+1}]$ if and only if the time window of $v_j$ in instance $I'$ includes the interval $[t'_i, t'_{i+1}]$. Furthermore, we will use the fact that on a directed cycle, for any vertices $u, v$, the travel time along the cycle direction satisfies $d(u, v) \leq C$. In particular, within time $2C$ one can traverse the entire cycle once (time $C$) and then still reach any target vertex in at most another $C$ time.

Next, we show that OPT = OPT′. First, let $S$ be an optimal solution of $I$ with profit OPT, we will show how to construct a feasible solution $S'$ of $I'$ with profit at least OPT. Furthermore, let $v_{S,t_i}, v_{S,t_{i+1}}$ be sequantial vertices visited in $S$ where profit is collected at time $t_i$ and $t_{i+1}$, respectively. For each consecutive pair of events $[t'_i, t'_{i+1}]$ for $0 \leq i \leq 2n - 1$, if $t_{i+1} - t_i \leq 2C$, the solution $S'$ follows exactly the same movement and waiting pattern as $S$ during $[t_i, t_{i+1}]$, with all times translated by $t'_i - t_i$. Observe that any vertex which has been collected in $S$ between $[t_i, t_{i+1}]$ will therefore be visited and collected between $[t'_i, t'_{i+1}]$ in $S'$.

Next, we consider event gaps that were shortened by the construction, i.e. those with $t_{i+1} - t_i > 2C$. Then $S'$ starts at $v_{S,t_i}$, traverses the cycle once to return to $v_{S,t_i}$, and then continues to $v_{S,t_{i+1}}$, where it waits until time $t'_{i+1}$. Note that every vertex is visited at least once in $[t'_i, t'_{i+1}]$ and thus every vertex collected by $S$ in $[t_i, t_{i+1}]$ is collected by $S'$ in $[t'_i, t'_{i+1}]$. It follows that $S'$ collects every vertex collected by $S$. Since the profit function is unchanged, therefore OPT′ ≥ OPT.

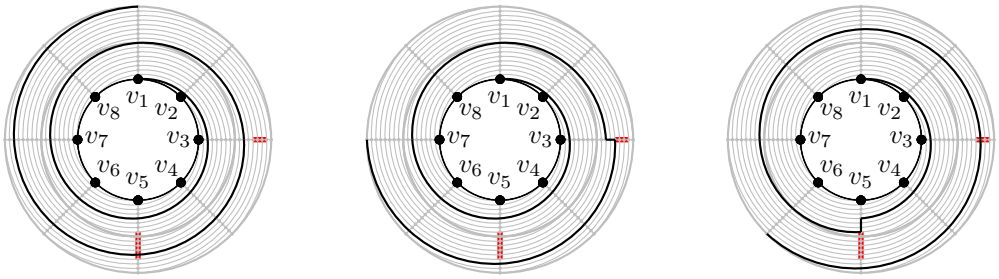
For the converse direction, let $S'$ be an optimal solution of $I'$. For each $i$, let $S$ follow the same movement as $S'$ during the time interval $[t'_i, t'_{i+1}]$. If $t_{i+1} - t_i > t'_{i+1} - t'_i$, then $S$ waits at the reached vertex for the remaining time so that it ends at time $t_{i+1}$. By doing so, each vertex collected in $[t'_i, t'_{i+1}]$ is visited and collected in $[t_i, t_{i+1}]$. Hence, $S$ collects every vertex collected by $S'$, and thus OPT ≥ OPT′.

Both constructions preserve every collected vertex. In particular, a walk collecting all vertex profits exists in $I$ if and only if such a walk exists in $I'$. Hence, $I$ is a yes-instance of COP-1TW if and only if $I'$ is a yes-instance. □

Using Lemma 3 the following theorem can be proven.

**Theorem 4.** *Given an instance of the COP-1TW on a directed cycle $G$ with arbitrary edge costs, the COP-1TW can be solved in $\mathcal{O}(n^4)$ time, where $n = |V(G)|$.*

*Proof.* W.l.o.g. by Lemma 3, assume that the latest deadline $d_{max} \leq 4nC$. We initialize a *schedule* $T$ containing for every $1 \leq i \leq n$ and $1 \leq j \leq 4n + 1$ a value $T_{i,j}$. Time $T_{i,j}$ is the




(a) The walk $W_T$ directly after initialization of schedule $T$ which does not wait.

(b) Vertex $v_3$ is not collected, hence time $T_{3,2}$ was increased to $r_3 = 12$ and the rest of the schedule is updated sequentially.

(c) Now vertex $v_5$ is no longer collected, hence time $T_{5,1}$ was increased to $r_5 = 7$.

Figure 2.1: An example of two iterations of the algorithm described in Theorem 4. The temporal graphs illustrate possible walks on a clockwise directed cycle with $n = 8$ and unit edge costs. The grey edges cannot be traversed in counterclockwise order and time windows are shown as red intervals on the radial edges.

time at which the corresponding walk $W_T$ arrives at vertex $v_i$ for the $j$th time. Moreover, there is a natural order on the tuples: we order pairs $(i, j)$ by increasing $j$, breaking ties by increasing $i$. We write $(i', j') \lhd (i, j)$ if it precedes $(i, j)$ in this order.

Initially, $T$ is a schedule, where $W_T$ never waits, i.e. $T_{i,j} := d(v_1, v_i) + C \cdot (j-1)$ (see Figure 2.1a). We call $T$ a *valid* schedule if $W_T$ starts at $v_1$ at time 0 and *hits* all vertices. We say that $W_T$ hits vertex $v_i$ if there exists $j \in [4n]$ with $T_{i,j} \in [r_i, d_i]$. Moreover, $W_T$ is called *proper* if the walk is monotone with respect to time and always either waits at vertices or follows the direction of the cycle.

We will modify $T$ iteratively until it is either valid or a conflict occurs, meaning there is no valid schedule. We apply the following modification iteratively: First, schedule $T$ is tested to see if it is valid. If $T$ is valid, $W_T$ can be returned, otherwise the algorithm continues with the following steps. If there exists a vertex $v_i$ with time window $[r_i, d_i]$ that walk $W_T$ does not hit, then we first check if $T_{i,1} > d_i$. If $T_{i,1} > d_i$, then we return that no valid schedule exists. Otherwise,

$$j := \max\{j \in [4n] \mid T_{i,j} < r_i\} \quad \text{and} \tag{2.2}$$

$$T_{i,j} := r_i. \tag{2.3}$$

Since $T_{i,j} < r_i$, this update strictly increases the scheduled visit time $T_{i,j}$. It can be realized by letting $W_T$ wait at $v_i$ during its $j$th visit until time $r_i$, and therefore $W_T$ remains proper.

Further, for all $T_{i',j'}$ with $(i, j) \lhd (i', j')$ set

$$T_{i',j'} := \max(T_{i',j'}, r_i + d(v_i, v_{i'}) + C \cdot (j' - j - 1[i' < i])). \tag{2.4}$$

Then, we start again at the beginning of this loop and test if there exists a vertex with a time window that is not currently hit. Two iterations of the algorithm are illustrated in Figure 2.1.

**Correctness of the algorithm.** To prove correctness, we show that the following loop invariant is maintained: walk $W_T$ is a proper walk on the cycle and, for each $1 \le i \le n$ and $1 \le j \le 4n$, $T_{i,j} \le S_{i,j}$ for any valid schedule $S$. In the first round of the algorithm, the

invariant trivially holds; walk $W_T$ is proper by definition and does not wait, hence for any valid schedule $S$, for all $i$ and $j$, $T_{i,j} \leq S_{i,j}$.

Assume that the loop invariant holds at the beginning of an iteration and let there be some $v_i$ for which $T_{i,1} > d_i$. Since the loop invariant is true we know that $d_i < T_{i,1} \leq S_{i,1}$ for any valid schedule. This inequality is only possible if no valid schedule exists, as it directly contradicts the definition of a valid schedule. Thus, the algorithm is correct if it returns that there is no valid schedule.

Next, we show that the properness of $W_T$ is preserved after the update in Equation 2.3 and the subsequent updates for all $(i,j) \lhd (i',j')$. The schedule remains unchanged for all $(i',j') \lhd (i,j)$, thus up to time $T_{i,j}$ the path remains proper. Then, updating $T_{i,j}$ is a safe operation as $r_i > T_{i,j}$, which means that $W_T$ will simply wait at vertex $v_i$ in the $j$th round. Furthermore, for all $(i,j) \lhd (i',j')$ the last visit times are updated based on the minimum travel time from $v_i$ to $v_{i'}$ with $j' - j$ rounds in between. Thus, $W_T$ remains proper since either $T_{i',j'}$ is the minimum travel time from leaving $v_i$ at $r_i$ or a larger value. Since all updates only increase visit times and each updated entry is set to at least the minimum travel time from $(v_i, r_i)$ plus $(j' - j)$ full rounds, the resulting walk remains monotone in time and moves only along the cycle direction (with possible waiting), hence the walk is still proper.

Finally, we show that the lower-bound invariant is preserved after the update step. Let $T'$ be the schedule after $T$ has been updated, we need to show that after an iteration the loop invariant holds for schedule $T'$ assuming the invariant holds for $T$. Trivially, the loop invariant holds for the schedule up to $T'_{i,j}$, since this part has not been modified. By maximality of $j$, we have $T_{i,j} < r_i \leq T_{i,j+1}$; since $v_i$ is not hit, it follows that $d_i < T_{i,j+1} \leq S_{i,j+1}$. So for any valid schedule, vertex $v_i$ must be collected strictly before the $j+1$th round, thus $S_{i,j} \geq r_i = T'_{i,j}$. Then, for all $(i,j) \lhd (i',j')$, where $r_i + d(v_i, v_{i'}) + C \cdot (j' - j - 1[i' < i]) > T_{i',j'}$, since $S$ corresponds to a proper walk $W_S$, $S_{i',j'} \geq r_i + d(v_i, v_{i'}) + C \cdot (j' - j - 1[i' < i]) = T'_{i',j'}$. Finally, for all $(i,j) \lhd (i'',j'')$, where $r_i + d(v_i, v_{i''}) + C \cdot (j'' - j - 1[i'' < i]) \leq T_{i'',j''}$, it holds that $T_{i'',j''} = T'_{i'',j''}$ and hence $S_{i'',j''} \geq T'_{i'',j''}$.

Thus, by our loop invariant, termination occurs only if no valid schedule exists or a valid schedule is found. Due to the choice of $j$ in Equation 2.2, for every $1 \leq j \leq 4n$, the value of $T_{i,j}$ can be updated at most once in Equation 2.3. Thus, after $\mathcal{O}(n^2)$ iterations, the algorithm must either report a valid schedule or that no such schedule exists. In each iteration, checking validity scans all $T_{i,j}$ and the update may propagate to all later entries, which is $O(n \cdot 4n) = O(n^2)$ time. Thus, a valid schedule will be computed in $\mathcal{O}(n^4)$ time if it exists. □

Thus, COP-1TW can be solved in polynomial time. In Section 2.1.2, we will see that the version with multiple time windows, COP-MTW, is NP-hard. The hardness of OP-1TW remains open for future work. However, in this section, we first give an exact algorithm for short windows, and then use it for a 2-approximation and an FPT algorithm.

**Theorem 5.** *Given an instance of the OP-1TW on a directed cycle $G$ with arbitrary profits and edge costs, and where every time window is shorter than the total length of the cycle, the OP-1TW can be solved in $\mathcal{O}(n^2 \log n)$ time.*

*Proof.* Let $C$ be the cost of traversing cycle $G$ exactly once. W.l.o.g. assume that $d_{max} \leq 4nC$ (Lemma 3). To solve the Orienteering Problem on a directed cycle with 'short' time windows the cycle can be unwrapped and concatenated to itself $4n$ times, resulting in a directed path $G'$. On this directed path the result of Proposition 2 can be applied.

More formally, for $1 \leq j \leq 4n$ and for each $1 \leq i \leq n$ create a vertex $v_{i,j}$ and add this to $V(G')$ with profit $\pi(v_i)$ and time window $[r_i, d_i]$, where $v_i \in V(G)$. Furthermore, for $1 \leq j \leq 4n$ and $1 \leq i < n$, add an edge $(v_{i,j}, v_{i+1,j})$ to $E(G')$ with the same cost as $(v_i, v_{i+1}) \in E(G)$. Finally, for $1 \leq j < 4n$, add an edge $(v_{n,j}, v_{1,j+1})$ to $E(G')$ with the cost of $(v_n, v_1) \in E(G)$. Let OPT be the profit of an optimal walk $W$ in $G$ and OPT′ be the profit of an optimal walk $W'$ in $G'$. Then it always holds that OPT = OPT′.

Since all time windows are shorter than $C$, walk $W'$ could not have collected two distinct vertices $v_{i,j}$ and $v_{i,j'}$ for some $1 \leq i \leq n$ and $1 \leq j < j' \leq 4n$. Therefore, any walk in $G$ can be modified to collect the same profit in $G'$ and vice versa. Hence, Proposition 2 can be used to solve $G'$, which can directly be used as a solution for $G$. Since $G'$ has $4n^2$ time windows, the solution to the OP-1TW on cycles with no time windows longer than or equal to $C$ can be calculated in $\mathcal{O}(n^2 \log n)$. □

The problem why a dynamic programming algorithm like the one in Theorem 5 does not extend to 'long' intervals is that when the algorithm encounters such an interval, it does not know whether it already collected the profit in a previous iteration. If all but $k$ intervals are 'short', the algorithm could of course, at the cost of a factor of $2^k$ in the runtime, keep track of the profit depending on which subset of the $k$ 'long' intervals it already collected. Alternatively, we can consider 'short' and 'long' intervals separately, resulting in a 2-approximation.

**Corollary 6.** *Given an instance of the OP-1TW on a directed cycle with arbitrary profits and edge costs, the OP-1TW is fixed-parameter tractable in the number of vertices whose time window has length at least $C$.*

**Corollary 7.** *Given an instance of the OP-1TW on a directed cycle $G$ with arbitrary profits and edge costs, there is a 2-approximation algorithm that runs in $\mathcal{O}(n^2 \log n)$ time.*

*Proof.* Let $C$ be the total cycle length of $G$. Furthermore, let $G_s$ and $G_l$ be exact copies of $G$. We will modify $G_s$ such that it only has 'short' time windows. More specifically, for $1 \leq i \leq n$ if $d_i - r_i \geq C$, in $G_s$ set $r_i = d_i = 0$ and $\pi(v_i) = 0$. Furthermore, we will modify $G_l$ such that it only has 'long' time windows. Analogously, for $1 \leq i \leq n$ if $d_i - r_i < C$, in $G_l$ set profit to $r_i = d_i = 0$ and $\pi(v_i) = 0$.

Using Theorem 5, we can solve the instance corresponding to $G_s$ obtaining profit $\mathrm{OPT}_s$. Furthermore, since all vertices in $G_l$ with profit have time windows longer than $C$, a walk which continues until $d_{max}$ collects all available profit $\mathrm{OPT}_l$.

Now let OPT be the profit of an optimal walk in $G$, then clearly $\mathrm{OPT} \leq \mathrm{OPT}_s + \mathrm{OPT}_l \leq 2 \max(\mathrm{OPT}_s, \mathrm{OPT}_l)$. Hence, $\max(\mathrm{OPT}_s, \mathrm{OPT}_l)$ is a 2-approximation that can be computed in $\mathcal{O}(n^2 \log n)$ time. □

In the following, we develop a polynomial-time approximation scheme for the OP-1TW. For this, we first consider (sub-)problems that can be optimally solved by a walk taking at most $k$ rounds. A *round* is a full traversal of the cycle starting and ending at $v_1$. The following lemma makes use of the generalization of the dynamic program as used in Proposition 2 for directed paths.

**Lemma 8.** *Given an instance of the OP-1TW on a directed cycle $G$ with arbitrary profits and edge costs where a walk can take at most $k$ rounds, the OP-1TW can be solved in $n^{\mathcal{O}(k)}$ time.*

*Proof.* Let $s_i$ denote the start time of round $i$ at $v_1$. Define the candidate set

$$S := \{0\} \cup \{\, t - d(v_1, v) \mid v \in V(G),\ t \in \{r_v, d_v\} \,\}.$$

Whenever $s_i$ is shifted within an interval between two consecutive values of $S$, then for every vertex $v$ the value $s_i + d(v_1, v)$ does not cross $r_v$ or $d_v$, therefore the relative position of the visit time w.r.t. $[r_v, d_v]$ is unchanged. Thus the set of profits that can be collected in round $i$ is unchanged, and we may move $s_i$ to a boundary point. Thus there exists an optimal solution with $s_1, \dots, s_k \in S$.

Since $|S| = \mathcal{O}(n)$, we enumerate all nondecreasing $k$-tuples $(s_1, \dots, s_k) \in S^k$, which are $n^{\mathcal{O}(k)}$ many. For a fixed choice of $(s_1, \dots, s_k)$, we use the same dynamic-programming idea as in Proposition 2, but with $k$ time parameters: a DP entry for vertex index $j$ stores the best profit attainable for $v_1, \dots, v_j$ together with the arrival times $t_1, \dots, t_k$ at $v_j$ in rounds $1, \dots, k$.

As in Proposition 2, we may assume that the walk only waits to hit a release time at a vertex whose profit is collected. Consequently, for each $j$ and each round $i$, it suffices to consider arrival times from the set

$$T_j := \{\, s_i + d(v_1, v_j) \mid i \in [k] \,\} \cup \{\, r_{v_\ell} + d(v_\ell, v_j) \mid \ell \le j \,\},$$

which has size $\mathcal{O}(n)$. Hence the DP has at most

$$n \cdot |T_j|^k = n^{\mathcal{O}(k)}$$

states, and each state can be updated from the previous layer by trying whether to collect $v_{j+1}$ in one of the $k$ rounds (or simply skip it), in time polynomial in $n$ and $k$. Therefore, for each guessed $(s_1, \dots, s_k)$ the DP runs in $n^{\mathcal{O}(k)}$ time, and multiplying by the $n^{\mathcal{O}(k)}$ guesses yields an overall runtime of $n^{\mathcal{O}(k)}$. □

Now suppose that we would compute for every pair of times $t < t'$ the maximum-profit walk that takes at most $k$ rounds. Unfortunately, concatenating such walks is not optimal, since they might collect the profit of the same vertices. To circumvent this problem we restrict to walks in which a *sprint* occurs occasionally, that is, a round in which the walk does not wait at vertices. A sprint takes exactly $C$ time.

We define a *$q$-sprint* as a sequence of at most $q$ rounds of which the last round is a sprint. Given two times $t < t'$ we can compute the maximum-profit *$k$-sprint* between these times similarly to Lemma 8, and these can be optimally concatenated using dynamic programming. The overall result is not a maximum profit walk but by choosing $k$ suitably depending on $\varepsilon$ we obtain a $(1+\varepsilon)$-approximation. Furthermore, we define a *$q$-workout* as a walk in which among any $q$ consecutive rounds, at least one round is a sprint. Equivalently, the number of non-sprint rounds between two consecutive sprints is at most $q - 1$.

**Lemma 9.** *Given an instance of the OP-1TW on a directed cycle with arbitrary profits and edge costs, the optimal $(2k-1)$-workout is a $k/(k-1)$-approximation of an optimal walk.*

*Proof.* Consider an optimal walk $W$ collecting OPT profit and split $W$ into sections of $k$ rounds. In each of these sections, replace the least profitable round by a sprint plus waiting time, giving us walk $W'$. Then, the maximum number of rounds between two sprints in $W'$ is $2k-2$, which means that $W'$ is a $(2k-1)$-workout. Furthermore, in the worst case, the profit is reduced at most by a factor of $(k-1)/k$. □

**Lemma 10.** *Given an instance of the OP-1TW on a directed cycle with arbitrary profits and edge costs, the optimal* $(2k-1)$*-workout can be computed in* $n^{\mathcal{O}(k)}$ *time.*

*Proof.* For every pair of times $t < t'$ we will compute the following: the optimal $(2k-1)$-sprint starting at $t$ and ending at $t'$, but ignoring profits of vertices where (a) time windows don't intersect $[t, t']$ or (b) the profit would have been collected by a sprint ending at time $t$.

This optimal $(2k-1)$-sprint can be computed with the algorithm of Lemma 8 by limiting the overall time to $[t, t']$ and the last round to a sprint (which can simply be achieved by first computing the profits collected by the sprint and removing those profits from the instance).

We now consider a directed graph with edges $(t, t')$ and as edge costs the profit of the optimal $(2k-1)$-sprint for $t, t'$ as computed above. Then the weight of the maximum-weight walk is the profit of the optimal $(2k-1)$-workout. For this, two conditions must be checked: Firstly, no profit is collected twice. Secondly, by not considering all profits when computing the profits of the $(2k-1)$-sprints, no profits were missed.

Assume some profit of vertex $v$ is collected more than once. Consider any but the first $(2k-1)$-sprint during which this profit is collected. Since there is a previous $(2k-1)$-sprint that collects the profit of $v$, the time window of $v$ spans back far enough so that it would have been collected by a sprint just before the $(2k-1)$ sprint. However, according to (b) above, the profit would have been excluded from consideration. Hence, profits are collected at most once.

Now consider an optimal $(2k-1)$-workout $W^\star$. Cut $W^\star$ at the end of each sprint round. This partitions $W^\star$ into consecutive segments, each ending with a sprint. By definition of a $(2k-1)$-workout, between two consecutive sprints there are at most $2k-2$ non-sprint rounds, hence each segment contains at most $(2k-1)$ rounds in total. Therefore, each segment is a $(2k-1)$-sprint, and the concatenation of these segments corresponds to a walk in the auxiliary time graph. For every edge $(t, t')$ in this walk, the edge weight equals the profit obtainable by the corresponding $(2k-1)$-sprint segment (after excluding profits as in (a) and (b)), so the maximum-weight walk in the time graph yields the optimal $(2k-1)$-workout profit. □

**Theorem 11.** *Given an instance of the OP-1TW on a directed cycle* $G$ *with arbitrary profits and edge costs, there is a* $(1+\varepsilon)$*-approximation algorithm that runs in* $n^{\mathcal{O}(1/\varepsilon)}$ *time.*

*Proof.* The theorem follows directly from Lemmas 9 and 10 by selecting $k - 1 = \lceil 1/\varepsilon \rceil$. □

### Multiple Time Windows

We show that the COP-MTW on directed cycles is NP-complete, implying NP-completeness for the OP-MTW.

**Theorem 12.** *The COP-MTW on directed cycles is NP-complete.*

*Proof.* The problem lies in NP: a certificate is a permutation of the vertices specifying the order in which they are first collected. Given such an order, we can verify feasibility in polynomial time by simulating the earliest possible walk that follows the directed cycle, waiting at each vertex only until the first time window that can be met (if any). If this

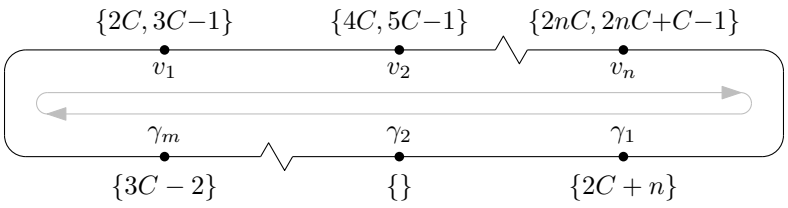



Figure 2.2: Example reduction from 3SAT, where $v_1$ appears in $\gamma_1$ as a positive and in $\gamma_m$ as a negated literal.

earliest simulation fails to collect some vertex in its order, then no walk can collect all vertices in that order since any additional waiting would only delay future arrivals.

In the following, we show the NP-hardness by a reduction from the well-known NP-complete problem 3SAT. Consider a 3SAT instance with a set $X$ of $n$ variables and a set $\Gamma$ of $m$ clauses. In 3SAT, each clause $\gamma \in \Gamma$ has exactly three literals, and the goal is to find an assignment of $X$ such that the formula is satisfied. We reduce this problem to the COP-MTW on a directed cycle with $N := n + m$ vertices and unit edge cost, hence one lap takes $N$ time. Since in this proof for the time windows only discrete points instead of intervals are needed, we use degenerate intervals $[t, t]$ to represent an admissible time point $t$.

For every $1 \le i \le n$, each variable $x_i \in X$ corresponds to $v_i$ in the cycle with time points $\{2iN, 2iN + N - 1\}$. Furthermore, for every $1 \le j \le m$, clause $\gamma_j \in \Gamma$ corresponds to vertex $v_{n+j}$ with an initial empty time window $\{\}$ (see Figure 2.2). Then, for every clause $\gamma_j$ in which $x_i$ appears as a positive literal, the time point $2iN + (n + j - i)$ is added to vertex $v_{n+j}$. Otherwise, if $x_i$ appears as a negated literal in $\gamma_j$, the time point $2iN + N - (m - j + i) - 1$ is added to $v_{n+j}$ (see Figure 2.2 for an example). Note that in this construction, no profit can be collected at time $t \in [iN, 2iN)$ for any $0 \le i \le n$.

We will now show that in the time range $[2iN, 3iN)$ it is not possible to collect the vertices corresponding to the variable $x_i$ and clauses $\gamma_j, \gamma_{j'}$ where $x_i$ appears as a positive and negative literal respectively. First of all, to collect $x_i$ we must be at vertex $v_i$ at time $2iN$ or $2iN + N - 1$. In the case that the $v_i$ is collected at time $2iN$, then it is possible to collect the vertex corresponding to $\gamma_j$ ($v_{n+j}$), since the distance between these vertices is exactly $n + j - i$. However, in this case it is not possible to collect $v_{n+j'}$: On one side, $v_i$ must be collected before $v_{n+j'}$ as $2iN + N - (m - j' + i) - 1 \ge 2iN$. However, the $v_{n+j'}$th vertex cannot be collected after $v_i$ either, since the distance between these vertices is exactly $n + j' - i$. This means that the earliest time $v_{n+j'}$ can be reached is at $2iN + n + j' - i = 2iN + N - n - m + n + j' - i = 2iN + N - (m - j' + i) > 2iN + N - (m - j' + i) - 1$. Hence, if $v_i$ is collected at time $2iN$, only the vertices corresponding to clauses where $v_i$ appears as a positive literal can be collected.

In the case that $v_i$ is collected at time $2iN + N - 1$, then the vertex corresponding to $\gamma_{j'}$ can be collected at an earlier time, since the distance is exactly $m - j' + i$. Furthermore, analogous to the previous case, we know that $2iN + (n + j - i) \le 2iN + N - 1$, so $v_{n+j}$ must be collected before $v_i$. However, this distance is $m - j + i$ and $2iN + (n + j - i) + (m - j + i) = 2iN + N > 2iN + N - 1$. Therefore, it is never possible to collect the vertices corresponding to $v_i$, $\gamma_j$, and $\gamma_{j'}$ between the times of $2iN$ and $3iN$.

Thus, if there exists a feasible solution in the 3SAT instance, the assignment could be used to choose when to visit the vertices corresponding to the variables. Since every clause is satisfied, this means that in our instance every clause vertex can be reached from at least one vertex variable, giving a valid walk that collects all vertices. Analogously, if a valid walk exists in the COP-MTW instance, we can use the time of collecting the vertices

corresponding to the variables for an assignment of the 3SAT instance. □

## 2.2 The Orienteering Problem on Dynamic Graphs

In the Orienteering Problem with time windows, profits are only collectible in certain time windows assigned to the corresponding vertex. A similar extension of the Orienteering Problem is to add time windows to the edges and thus consider the Orienteering Problem in a setting with dynamic graphs. For an instance of the Orienteering Problem on dynamic graphs with input graph $G = (V, E)$, the time windows for edges are sets of intervals $\mathcal{I}_e$ for all $e \in E$. As before, we model time continuously, i.e., $t \in \mathbb{R}_{\geq 0}$. For each edge $e \in E$ we are given a set $\mathcal{I}_e$ of (pairwise disjoint) availability intervals. An edge $e$ is *active* at time $t$ if $t \in I$ for some $I \in \mathcal{I}_e$. A timed walk is a sequence $(v_0, t_0), (v_1, t_1), \dots, (v_\ell, t_\ell)$, with $v_0 = s$ and $t_0 = 0$ such that for each step there is an edge $e = (v_k, v_{k+1})$ and

$$t_{k+1} \geq t_k + c(e) \quad \text{and} \quad t_k \in \bigcup \mathcal{I}_e.$$

Thus, the edge needs to be only active at departure times. If $t_{k+1} > t_k + c(e)$, the walk waits at $v_k$ before traversing $e$. Furthermore, the total time must be less than the budget $B$, i.e. $t_\ell \leq B$.

In contrast to the Orienteering Problem with time windows, directed paths or cycles can be solved trivially by simply advancing the walk whenever the next edge becomes active. Surprisingly, even dynamic undirected paths can be solved with a simple algorithm, as shown in Proposition 13. However, the problem immediately becomes NP-hard when dynamic trees are considered, as shown in Proposition 14.

**Proposition 13.** *Given an instance of the Orienteering Problem with a dynamic undirected path $G$, this instance can be solved in $\mathcal{O}(n^2)$ time.*

*Proof.* Let $V(G) = \{v_1, \dots, v_n\}$ with $E(G) = \{\{v_i, v_{i+1}\} \mid 1 \leq i < n\}$, where $v_\sigma = s$. Let OPT be the profit collected by an optimal walk $W_{\text{OPT}}$ and $i^*$ and $j^*$ the left- and rightmost vertices visited in $W_{\text{OPT}}$. Formally,

$$i^* := \min\{i \mid v_i \in V(W_{\text{OPT}})\} \qquad \text{and} \qquad j^* := \max\{i \mid v_i \in V(W_{\text{OPT}})\}.$$

Thus, all vertices visited by $W_{\text{OPT}}$ lie in $\{v_{i^*}, \dots, v_{j^*}\}$ and hence $\text{OPT} = \sum_{k=i^*}^{j^*} \pi(v_k)$.

Let $t_i$ and $t_j$ be the first times at which $W_{\text{OPT}}$ visits $v_{i^*}$ and $v_{j^*}$, respectively, and assume w.l.o.g. that $t_i \leq t_j$. Since $G$ is a path, $W_{\text{OPT}}$ must traverse the unique subpath from $s$ to $v_{i^*}$ and later the unique subpath from $v_{i^*}$ to $v_{j^*}$ (possibly with detours and waiting). Consider the canonical walk $\hat{W}$ that follows these subpaths in this order and, whenever the next edge on the subpath is active, traverses it immediately (otherwise it waits until it becomes active). By construction, $\hat{W}$ never waits or detours unless forced by edge inactivity, hence it reaches $v_{i^*}$ no later than $t_i$ and $v_{j^*}$ no later than $t_j$. In particular, $\hat{W}$ is feasible within budget $B$ and visits all vertices $v_k$ with $i^* \leq k \leq j^*$, collecting profit $\sum_{k=i^*}^{j^*} \pi(v_k)$.

Therefore, an optimal solution is characterized by the leftmost (or rightmost) vertex $v_i$ reached first. For each $1 \leq i \leq n$ we simulate the canonical walk that starts at $s$, moves toward $v_i$ (traversing each edge as soon as it becomes active, waiting only if necessary), and then continues from $v_i$ in the opposite direction as long as the budget allows. Among

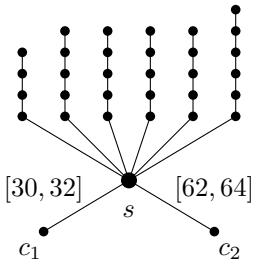



Figure 2.3: Example reduction for the instance $\{4, 5, 5, 5, 5, 6\}$, making $T = 15$. All upper edges are active at time $t \in [0, 2mT + 2m]$.

all choices of $i$, we return the walk of maximum collected profit. Thus, by guessing $v_i$ we obtain an optimal walk in $\mathcal{O}(n^2)$ time. □

**Proposition 14.** *The Orienteering Problem on dynamic trees is NP-hard.*

*Proof.* We will prove that the Orienteering Problem is NP-hard on dynamic trees by a reduction from the 3-partition problem. Given a multiset $I$ of $n = 3m$ integers, such that $T/4 < i < T/2$ for all $i \in I$ and with total sum $mT$, i.e. $\sum_{i \in I} i = mT$. Can $I$ be partitioned into a set of triplets $I_1, \cdots, I_m$ such that the sum of each triplet is exactly $T$?

Given such a 3-partition instance, we will create an Orienteering Problem instance on a dynamic graph $(V, E)$. The constructed graph $(V, E)$ will be a spider graph with start node $s$ at the center. Then, for each integer $i \in I$, we connect a leg of length $i$ to $s$, whose edges are active at times $[0, 2mT + 2m]$. Next, we add $m$ control vertices $c_1, \cdots, c_m$ with edges to $s$, such that the edge incident to $c_i$ is active at $[2iT + 2(i - 1), 2iT + 2(i - 1) + 1]$. Furthermore, we say each vertex has 1 profit, edge traversal costs are 1 and set the budget to $B := 2mT + 2m$. An example of such a graph can be seen in Figure 2.3.

What remains is to show that it is possible to collect exactly $m + mT + 1$ profit in our Orienteering Problem instance if and only if the 3-partition instance is a yes-instance. Assume there exists a walk $W$ that collects profit $m + mT + 1$, i.e., it visits every vertex at least once. Since each control vertex $c_i$ has only the single active interval $[2iT + 2(i - 1), 2iT + 2(i - 1) + 2]$ on its incident edge, $W$ must traverse $s \to c_i$ within this interval for every $i$. In particular, between two consecutive control visits there are exactly $2T$ time units available to traverse leg edges. Thus, in each phase $W$ must spend exactly $2T$ time on legs, which means it fully traverses a set of legs whose lengths sum to $T$ (full out-and-back traversal of a leg of length $\ell$ costs $2\ell$ time). By the 3-PARTITION promise $T/4 < \ell < T/2$, no phase can contain more than three legs, and since all $3m$ legs are visited in total, each phase contains exactly three legs summing to $T$. This yields a valid 3-partition.

Now suppose we can partition $I$ into triplets $I_1, \cdots, I_m$ such that the sum of each triplet is exactly $T$. Then there exists a walk in $(V, E)$, where we first visit the legs corresponding to the integers of $I_1$, this takes exactly $2T$ time and collects $T$ profit. Thereafter, we visit the first control vertex $c_1$, whose incident edge is active in interval $[2T, 2T + 1]$, and go back to $s$, resulting in 1 additional profit and 2 additional time. We repeat these steps for every interval $I_i$, each time collecting the corresponding legs and then visiting the $i$th control vertex. Therefore, this walk collects exactly $m + mT + 1$ profit. □

## 2.3 The Orienteering Problem without time windows

In general, the Orienteering Problem is NP-hard even on trees with arbitrary profits and thus also on graphs with bounded treewidth. If vertex profits are unit, the Orienteering Problem is still NP-hard on arbitrary graphs [13], but we show that even with vertex profits it is possible to give a dynamic program when the graph has bounded treewidth. We will extend this result to a $(1+\varepsilon)$-approximation for the Orienteering Problem with arbitrary vertex profits on graphs with bounded treewidth.

Note that it is quite easy to show that the Orienteering Problem with unit vertex profits is polynomial for a tree, by giving a dynamic program: For every vertex we only need to distinguish if a walk has visited the vertex an even or an odd number of times. In contrast, for graphs with bounded treewidth, we have to add an additional partition of special structures to the dynamic program.

**Lemma 15.** *Given an instance of the Orienteering Problem with unit vertex profit and input tree $G$, the Orienteering Problem is solvable in $\mathcal{O}(n^2)$ time.*

*Proof.* Root the tree at $s$. W.l.o.g. assume the rooted tree is binary: if a node has more than two children, we replace it by a small binary tree of auxiliary vertices of profit 0 connected by edges of cost 0, so that each original child becomes attached to a leaf of this auxiliary structure. This transformation preserves feasibility and the set of collectible profits, and increases the number of vertices only by $\mathcal{O}(n)$.

Let $\pi(v) \in \{0,1\}$ denote the profit of vertex $v$ (all original vertices have $\pi(v)=1$, auxiliary vertices have $\pi(v)=0$), and let $p(v)$ be the total profit contained in the subtree of $v$ (i.e., $p(v) = \sum_{u \in T_v} \pi(u)$). Then, we will fill a dynamic program $a[v, \Pi, d]$, where $v \in V$, $0 \le \Pi \le n$, and $d \in \{\text{CLOSED}, \text{OPEN}\}$. It stores the cost of a minimum-cost walk which starts at vertex $v$, visits exactly $\Pi$ vertices with profit 1, and whether the walk returns to $v$ (closed walk) or may end anywhere in the subtree (open walk).

Then, $a$ can be filled by processing the vertices in a topological order and the entries can be filled based on the type of $v$

**Leaf.** let $v$ be a leaf in $G$, then

$$a[v, \pi(v), \text{CLOSED}] = a[v, \pi(v), \text{OPEN}] = 0,$$

and all other entries for $v$ get value $+\infty$.

**Inner node.** let $v$ be an inner node in $G$, with children $w_1, w_2$, then

$$a[v, i, \text{CLOSED}] = \min \begin{Bmatrix} a[w_1, i-\pi(v), \text{CLOSED}] + 2c(v, w_1), \\ a[w_2, i-\pi(v), \text{CLOSED}] + 2c(v, w_2), \\ \min\limits_{1 \le j \le i-\pi(v)} a[w_1, j, \text{CLOSED}] + a[w_2, i-\pi(v)-j, \text{CLOSED}] + 2c(v, w_1) + 2c(v, w_2) \end{Bmatrix}$$

$$a[v,i,\text{OPEN}] = \min \left\{ \begin{array}{l} a[w_1, i-\pi(v), \text{OPEN}] + c(v,w_1), \\ a[w_2, i-\pi(v), \text{OPEN}] + c(v,w_2), \\ \min\limits_{1\le j\le i-\pi(v)} a[w_1,j,\text{OPEN}] + a[w_2, i-\pi(v)-j, \text{CLOSED}] + c(v,w_1) + 2c(v,w_2), \\ \min\limits_{1\le j\le i-\pi(v)} a[w_1,j,\text{CLOSED}] + a[w_2, i-\pi(v)-j, \text{OPEN}] + 2c(v,w_1) + c(v,w_2) \end{array} \right\}$$

**Runtime.** Fix an inner node $v$ with children $w_1, w_2$. Although the recurrences are written with expressions of the form $\min_j a[w_1, j, \cdot] + a[w_2, (i-\pi(v))-j, \cdot]$ (and analogously for OPEN), we do *not* evaluate them by looping over $j$ separately for every $i$. Instead, we compute *all* values $a[v,i,\text{CLOSED}]$ and $a[v,i,\text{OPEN}]$ by one double loop: for every $0 \le j \le p(w_1)$ and $0 \le \ell \le p(w_2)$ we set $i = \pi(v)+j+\ell$ and *relax* the corresponding entries, i.e. perform updates of the form

$$a[v,i,d] \leftarrow \min\{a[v,i,d],\ \text{candidate}(j,\ell,d)\},$$

where candidate$(j,\ell,d)$ is exactly the right-hand side cost implied by the recurrence (for example, for the term using two closed child-walks, $a[w_1,j,\text{CLOSED}] + a[w_2,\ell,\text{CLOSED}] + 2c(v,w_1) + 2c(v,w_2)$, and similarly for the three OPEN combinations). Thus, all values for node $v$ are computed in time $\mathcal{O}(p(w_1)\cdot p(w_2))$ (up to a constant factor).

Summing over all binary internal nodes yields $\mathcal{O}(n^2)$ total time: each unordered pair of profit-1 vertices $\{x,y\}$ is separated into different subtrees at exactly one node, namely at LCA$(x,y)$, so it contributes to $p(w_1)p(w_2)$ for exactly one node. Hence $\sum_v p(w_1)p(w_2) = \mathcal{O}(n^2)$, and the overall runtime is $\mathcal{O}(n^2)$.

□

**Theorem 16.** *Given an instance of the Orienteering Problem with unit vertex profits and input graph $G$ of bounded treewidth $k$, the Orienteering Problem is solvable in $\mathcal{O}(n^2)\cdot k^{\mathcal{O}(k)}$ time.*

*Proof.* W.l.o.g assume that $\mathcal{T} = (T, \{X_t\}_{t\in V(T)})$ is a nice tree decomposition for $G$ of width $k$ as given in [26]. Note that their definition of a "nice" tree-decomposition is slightly different from the commonly used one, which includes *introduce*, *forget*, or *join* node. In [26], the authors give *introduce edge* and *introduce vertex* nodes, where the introduce vertex nodes correspond to introduce nodes in the standard sense and an introduce edge node $t$ for edge $uv \in E(G)$ is labelled by $uv$, has exactly one child $t'$ such that $X_t = X_{t'}$ and it holds that $u, v \in X_t$. Furthermore, for every edge $uv \in E(G)$, there is exactly one introduce edge node labelled by $uv$ in $\mathcal{T}$.

Further assume that $s \in X_t\ \forall t \in V(T)$ (increases treewidth by at most 1) and $X_v = \{s\}$ if $v$ is the root or a leaf. Let $G_t$ for $t \in V(T)$ be the induced subgraph of $G$ with vertex set $\bigcup_{t'\ge t} X_{t'}$, that is, the subgraph of $G$ containing all vertices in the bags of the subtree of $T$ rooted at $t \in V(T)$.

For a walk $H$ starting at $s$, for some $t \in T$, the part of $H$ contained in $G_t$ is a non-empty set of disconnected walks. Let $F$ be such a set of disconnected walks that collects the most distinct vertices and let $X \subseteq X_t$ such that for any profit $1 \le \Pi \le n$ it holds that $s \in V(F)$, $V(F)\cap X_t = X$, $|V(F)| = \Pi$, and $P = \{X_t \cap C \mid C \text{ is a connected component of } V(F)\}$ is a partition of $X$, where $V(F)$ is the set of vertices in $F$.

We then define $c[t, X, P, \mathcal{D}, \Pi]$ as the minimum cost of a possible $F$ in $G_t$. In addition to the partition $\mathcal{P}$ of $X$, we store the set $\mathcal{D} \subseteq X$ of OPEN vertices, i.e., those vertices of odd degree in the multigraph induced by the partial solution inside $G_t$. Vertices in $X \setminus \mathcal{D}$ are CLOSED (even degree).

The best profit is obtained by finding the maximum $0 \leq \Pi \leq n$ such that $c[r, \{s\}, \{\{s\}\}, \mathcal{D}, \Pi] \leq B$, where $r \in \mathcal{T}$ such that $G_r = G$ and $\mathcal{D} = \emptyset$ or $\mathcal{D} = \{s\}$. If $F$ does not exist, set $c[t, X, P, \mathcal{D}, \Pi] = +\infty$. Otherwise, fill $c[t, X, P, \mathcal{D}, \Pi]$ depending on the node type of $t$:

**Leaf node.** Let $t$ be a leaf node. Then $X_t = \{s\}$, set $c[t, \{s\}, \{\{s\}\}, \emptyset, 1] = 0$ and all other entries for $t$ to $+\infty$.

**Introduce vertex node.** Let $t$ be an introduce vertex node with child node $t'$ s.t. $X_t = X_{t'} \cup \{v\}$, $v \notin X_{t'}$. As edges are introduced in introduce edge nodes, assume $v$ is isolated in $G_t$. Then for all $X \subseteq X_t$, partition $P = \{P_1, P_2, \cdots, P_q\}$ of $X$, and $0 \leq \Pi \leq n$: If $v$ is in $X$, then $\{v\} \in P$, that is $v$ is its own connected component. Thus:

$$c[t, X, P, \mathcal{D}, \Pi] = \begin{cases} c[t', X \setminus \{v\}, P \setminus \{\{v\}\}, \mathcal{D} \setminus \{v\}, \Pi - 1] & \text{if } v \in X \text{ and } v \in \mathcal{D}, \\ c[t', X \setminus \{v\}, P \setminus \{\{v\}\}, \mathcal{D}, \Pi - 1] & \text{if } v \in X \text{ and } v \notin \mathcal{D}, \\ c[t', X, P, \mathcal{D}, \Pi] & \text{otherwise.} \end{cases}$$

**Introduce edge node.** Let $t$ be an introduce edge node labelled by $uv$, and let $t'$ be its child (so $X_t = X_{t'}$). Consider every $X \subseteq X_t$, a partition $P$ of $X$, a set $\mathcal{D} \subseteq X$, and $0 \leq \Pi \leq n$. If $u \notin X$ or $v \notin X$, then the partial solution does not use $u$ or $v$ and the new edge is irrelevant, hence $c[t, X, P, \mathcal{D}, \Pi] = c[t', X, P, \mathcal{D}, \Pi]$.

Otherwise, we may either ignore the edge $uv$, traverse it exactly once, or traverse it exactly twice. Traversing $uv$ can only affect the connected component containing $u$ and $v$. If $u$ and $v$ lie in different blocks of $P$, then traversing $uv$ merges these two blocks. If they already lie in the same block, the partition $P$ stays unchanged.

$$c[t, X, P, \mathcal{D}, \Pi] = \min \left\{ \begin{array}{l} c[t', X, P, \mathcal{D}, \Pi] \\ \min_{P' \in P^*} c[t', X, P', \mathcal{D} \triangle \{u, v\}, \Pi] + c(uv), \\ \min_{P' \in P^*} c[t', X, P', \mathcal{D}, \Pi] + 2c(uv), \end{array} \right\},$$

where $\mathcal{D} \triangle \{u, v\}$ denotes symmetric difference, and $P^*$ is the set of partitions obtained from $P$ by merging the blocks containing $u$ and $v$ (if they are distinct). To ensure that each block in $P$ represents a closed walk or an open walk with two end points, we only consider terms for which the invariant $|P \cap \mathcal{D}| \in \{0, 2\}$ holds for every $P \in P$.

**Forget node.** Suppose that $t$ is a forget node with child node $t'$ such that $X_t = X_{t'} \setminus \{w\}$ for some $w \in X_{t'}$. Consider every $X \subseteq X_t$, a partition $P$ of $X$, a set $\mathcal{D} \subseteq X$, and $0 \leq \Pi \leq n$.

We may either not use $w$ (so the child state already equals $(X, P, \mathcal{D})$), or use $w$ and forget it. In the latter case we extend $(X, P)$ by inserting $w$ into some block of $P$ (or as a singleton

block), and we require that $w$ is *closed* at the boundary, i.e. $w \notin \mathcal{D}'$. Hence,

$$c[t, X, P, \mathcal{D}, \Pi] = \min\Big\{c[t', X, P, \mathcal{D}, \Pi], \min_{(X', P', \mathcal{D}') \in \mathcal{F}} c[t', X', P', \mathcal{D}', \Pi]\Big\},$$

where $\mathcal{F}$ contains all triples $(X', P', \mathcal{D}')$ with $X' = X \cup \{w\}$, $\mathcal{D}' = \mathcal{D}$, and $P'$ obtained from $P$ by adding $w$ to exactly one block, subject to $w \notin \mathcal{D}'$ and the invariant $|P \cap \mathcal{D}'| \in \{0, 2\}$ for all $P \in P'$.

**Join node.** Suppose $t$ is a join node with children $t_1$ and $t_2$. Consider for all $X \subseteq X_t$, a partition $P$ of $X$, a set $\mathcal{D} \subseteq X$, and $0 \leq \Pi \leq n$.

We combine two child states $(X, P_1, \mathcal{D}_1, \Pi_1)$ and $(X, P_2, \mathcal{D}_2, \Pi_2)$ as follows.

First, the open vertices must satisfy $\mathcal{D} = \mathcal{D}_1 \triangle \mathcal{D}_2$ (degrees add modulo 2). Second, the partition $P$ is the partition such that two vertices of $X$ lie in the same block of $P$ iff they are connected in at least one of the two child partial solutions. Finally, since the vertices in $X$ are counted in both subsolutions, the profits must satisfy $\Pi = \Pi_1 + \Pi_2 - |X|$. Hence,

$$c[t, X, P, \mathcal{D}, \Pi] = \min\big\{ c[t_1, X, P_1, \mathcal{D}_1, \Pi_1] + c[t_2, X, P_2, \mathcal{D}_2, \Pi_2] \big\},$$

where the minimum ranges over all $P_1, P_2, \mathcal{D}_1, \mathcal{D}_2, \Pi_1, \Pi_2$ satisfying $\text{join}(P_1, P_2) = P$, $\mathcal{D}_1 \triangle \mathcal{D}_2 = \mathcal{D}$, and $\Pi_1 + \Pi_2 - |X| = \Pi$.

This completes the computation of $c$. As every bag has size at most $k + 2$ and profit $0 \leq \Pi \leq n$, the number of states per node is at most $n \cdot 2^{k+2} \cdot (k+2)^{k+2} \cdot 2^{k+2} = n \cdot k^{\mathcal{O}(k)}$. Since $T$ has $\mathcal{O}(n)$ nodes, the total runtime of the algorithm is $\mathcal{O}(n^2) \cdot k^{\mathcal{O}(k)}$. □

While we mainly need the result of Theorem 16 to give a $(1+\varepsilon)$-approximation algorithm for the general Orienteering Problem on graphs with bounded treewidth, it also means that the Orienteering Problem with unit vertex profits is fixed-parameter tractable for the parameter treewidth, since a tree decomposition of a fixed width can be computed in linear time [14]. We note that the result of Theorem 16 was obtained independently in [70], where the authors study the P2P-Orienteering problem. This problem is very similar to the Orienteering Problem as we define it, but includes both a start point and an end point for the required walk, instead of only a start point, and only considers unit profits. They show that the P2P-Orienteering problem is fixed-parameter tractable on graphs with bounded treewidth by giving a different dynamic program from ours.

Using similar techniques as in [21, 36, 53], Theorem 16 can be extended to an approximation for graphs with bounded treewidth and arbitrary profits.

**Theorem 17.** *There exists a $(1+\varepsilon)$-approximation algorithm for the Orienteering Problem on graphs with bounded treewidth $k$ that takes $n^{\mathcal{O}(1)} \cdot k^{\mathcal{O}(k)} \cdot \text{poly}(1/\varepsilon)$ time.*

*Proof.* Guess $P = \pi_{\max}$, the maximum profit of a vertex visited by an optimal walk, by iterating over all values $P \in \{\pi(v) \mid v \in V\}$, deleting all vertices with profit $> P$, and taking the best result over all guesses. Fix a guess $P$ and set

$$K := \frac{\varepsilon P}{(1+\varepsilon)n}.$$

Define scaled integer profits $\tilde{\pi}(v) := \lfloor \pi(v)/K \rfloor$.

We run the treewidth dynamic program (Theorem 16) on the same graph, replacing the profit counter by $\tilde{\Pi} \in \{0, \dots, \sum_v \tilde{\pi}(v)\}$ and using $\tilde{\pi}$ as profits, which yields an optimal walk $\tilde{W}$ for the scaled instance.

For any walk $W$ visiting at most $n$ vertices we have $K\tilde{\pi}(W) \leq \pi(W) < K\tilde{\pi}(W) + nK$. Let $W^*$ be an optimal walk for the original instance under the correct guess $P$. By optimality of $\tilde{W}$ for scaled profits,

$$\pi(\tilde{W}) \;\geq\; K\tilde{\pi}(\tilde{W}) \;\geq\; K\tilde{\pi}(W^*) \;\geq\; \pi(W^*) - nK \;=\; \mathrm{OPT} - \frac{\varepsilon P}{1+\varepsilon} \;\geq\; \mathrm{OPT} - \frac{\varepsilon}{1+\varepsilon}\mathrm{OPT} \;=\; \frac{\mathrm{OPT}}{1+\varepsilon}.$$

Hence $\tilde{W}$ is a $(1+\varepsilon)$-approximate solution.

Finally, since all vertices with profit greater than $P$ were deleted, we have $\pi(v) \leq P$ for every remaining vertex. Hence, $\tilde{\pi}(v) \leq P/K$, and therefore $\sum_v \tilde{\pi}(v) = \mathcal{O}(n^2/\varepsilon)$. Thus, the runtime is $n^{\mathcal{O}(1)} \cdot k^{\mathcal{O}(k)} \cdot \mathrm{poly}(1/\varepsilon)$, including the $O(n)$ guesses for $P$. □

Note that we cannot expect to find an exact polynomial-time algorithm for the Orienteering Problem on graphs with bounded treewidth (unless P=NP), since it is already NP-hard for $k = 1$, i.e. on trees.

**Proposition 18.** *The Orienteering Problem is NP-hard even on trees.*

*Proof.* We prove NP-hardness by a reduction from the Knapsack problem. Recall the decision variant of the Knapsack problem: Given a set of $n$ items with sizes $s_1, s_2, \cdots, s_n$, values $p_1, p_2, \cdots, p_n$, a capacity $K$, and a target value $P$, does a subset $S \subseteq \{1, 2, \cdots, n\}$ exist, such that $\sum_{i \in S} s_i \leq K$ and $\sum_{i \in S} p_i \geq P$?

Given such an instance, construct an Orienteering Problem instance $((V, E), \pi, c, s, B)$. Let $s$ be the start vertex with $\pi(s) = 0$. For each item $i$, add a vertex $v_i$ with profit $\pi(v_i) = p_i$, add an edge $e_i = (s, v_i)$ to $E$ with cost $c(e_i) = s_i/2$. Furthermore, since a walk can end in any vertex, we add one more vertex $v_E$ whose profit is larger than the total profit of all item vertices, and connect it to $s$ by an edge $(s, v_E)$ with cost $c((s, v_E)) = 2K$. Finally, we set $B = 3K$. The graph of our constructed instance is a tree. A walk that collects the maximum profit on this instance would immediately give an optimal solution to the corresponding knapsack instance. □

# Chapter 3
# The Touring Problem

A common challenge when planning an outdoor activity is finding an appropriate route and modeling this as the Orienteering Problem (see Chapter 2) might lack realism. Depending on the activity from hiking and running to gravel and road cycling, the requirements for a desired route can vary considerably. More specifically, instead of having profit on vertices, one might want to assign profit on edges, e.g., when the profit is related to the scenery or the road quality.

This problem can be modeled as the *Edge Orienteering Problem* (EOP), which is the undirected version of the Arc Orienteering Problem (AOP). More formally, the EOP is defined as follows: given a quintuple $(G = (V, E), s, \pi, c, B)$, where $G$ is a simple undirected graph with $n = |V|$ vertices, $s \in V$ is the starting point, $B \in \mathbb{R}_{>0}$ represents the available budget, and $\pi : E \to \mathbb{R}_{\geq 0}$ and $c : E \to \mathbb{R}_{\geq 0}$ are the profit and cost functions, respectively. The objective of the EOP is to find a (not necessarily simple) closed walk $P = (e_1, \cdots, e_\ell)$ starting and ending at $s$ such that

$$\sum_{i=1}^{\ell} c(e_i) \leq B \text{ and}$$

$$\sum_{e \in E(P)} \pi(e) \text{ is maximized,}$$

where $E(P)$ denotes the set of (undirected) edges traversed by $P$.

In other words, we are trying to find a walk in $G$ that does not exceed the budget $B$ and collects as much profit as possible. Edges are allowed to be used multiple times; however, their profit is only counted once. Note that occasionally in literature the problem asks for a closed walk that need not include a specific vertex [36]. Given an algorithm that solves an instance with a start vertex, instances without a specific vertex can be solved by simply guessing the start vertex, which would not impact the approximation factor.

The best known approximation for the EOP is a $(6 + \varepsilon + o(1))$-approximation[1] given by Gavalas, Konstantopoulos, Mastakas, Pantziou, and Vathis [36], for any $\varepsilon > 0$. The contribution of this chapter is to improve this approximation factor by presenting a $(4 + \varepsilon)$-approximation.

Similar to the EOP, the AOP on directed graphs has been considered by Gavalas et al. Here a simple $\mathcal{O}(\frac{\log^2 m}{\log \log m})$-approximation for the AOP is presented, where $m$ is the number of edges. When the profits are given on vertices rather than edges, the problem is

[1] The $o(1)$-factor here could be hidden in the $\varepsilon$-factor by handling small instances using an exact algorithm, thus this algorithm could also be seen as a $(6 + \varepsilon)$-approximation.

simply called the Orienteering Problem. Chekuri, Korula, and Pál [21] presented a $(2+\varepsilon)$-approximation for the Orienteering Problem, improving on previous work [13, 9]. Furthermore, both the AOP and Orienteering Problem are APX-hard in undirected graphs, the best known inapproximability ratio is at most 1481/1480 [21, 36].

While there is only a small amount of work on the theoretical side of the Arc and Edge Orienteering problems, there exists a large body of research on the practical side, motivated by its application in route planning. Vansteenwegen, Souffriau, and Van Oudheusden [90] proposed a branch-and-cut algorithm that solves the AOP exactly along with an iterative local search metaheuristic that can solve larger instances. Lu and Shahabi [59] followed up by introducing spatial database techniques to solve even larger instances.

Closely related is the generation of tours for jogging and cycling [38, 76]. These papers solve a problem analogous to the EOP by applying a min-cost heuristic. The approach used there is to enhance the cost of edges by reducing the cost of an edge if that edge is profitable. Then, the task is to find intermediate points between which the min-cost heuristic could be applied. An overview and demonstrations of these approaches are given in [36].

## 3.1 Approximating the Edge Orienteering Problem

In this section we present a $(4+\varepsilon)$-approximation for the EOP for any $\varepsilon > 0$. Without loss of generality, we assume that for every edge $e = \{u, v\} \in E$ there exists a closed walk $P$ starting and ending at $s$ such that $P$ traverses $e$ at least once and has total cost at most $B$. Any edge that does not satisfy this property can be removed, since it cannot appear in any feasible solution.

Like Gavalas et al. [36], we reduce to the Orienteering Problem with unit profit per vertex. Since in the EOP profits are not necessarily polynomially bounded, we need the following lemma.

**Lemma 19** ([36])**.** *An $\alpha$-approximation algorithm for the EOP with polynomially bounded positive integer profits yields an $(\alpha + o(1))$-approximation algorithm for the EOP.*

Since an $\varepsilon$ factor is introduced later, we choose to hide this $o(1)$ factor in $\varepsilon$. Gavalas et al. show that the $o(1)$-factor can be bounded by $1/n$. Hence, we can simply solve an instance by brute force when $1/n > \varepsilon$ and otherwise use the reduction of Lemma 19.

Conceptually, Gavalas et al. reduce to the Orienteering Problem by splitting each edge and placing its profit in the middle. Thus, if both split edges are traversed, the profit is collected for the cost of the original edge as intended. However, a walk can also go to the midpoint and return, collecting the full profit while paying only one edge traversal. In the original problem, returning to the start side after collecting the profit would cost in total two traversals, so in the reduction one gets the same profit for half the cost.

In contrast, our approach can be thought of as distributing the profit more evenly, so that the profit earned is roughly proportional to how much of the edge has been traversed. The advantage of this is that if, e.g., the walk goes to the middle of the edge and back, it also only gains half the profit. The disadvantage of our approach is that it is much easier to gain partial profits at low cost. We show how to mitigate this disadvantage so that we obtain a $(4+\varepsilon)$-approximation instead of the $(6+\varepsilon+o(1))$-approximation of Gavalas et al.

To formalize the idea of the previous paragraph and to achieve the $(4+\varepsilon)$-approximation,

we reduce to an intermediate problem before reducing to the Orienteering Problem: the *Continuous Edge Orienteering Problem* (CEOP).

The CEOP is similar to the EOP except that we are allowed to turn back at any point on an edge. To allow for walks to return at any point, we define a *partial edge* as $(\delta, e)$, where $0 < \delta \leq 1$ and $e \in E$. Here $\delta$ indicates how much of the edge is taken from a walk. If $\delta < 1$, the walk starts at one of the endpoints $v$ and travels along the edge $e$, collecting $\delta\pi(e)$ profit at a cost of $2\delta c(e)$ and ends back at endpoint $v$ (Figure 3.1). If the walk uses a complete edge, we denote this by $\delta = 1$. In this case, however, the walk goes from one of the endpoints to the other (and not back unless the edge is included a second time in the walk). Thus, we introduce a new cost function that can handle partial edges:

$$c'(\delta, e) = \begin{cases} c(e) & \text{if } \delta = 1, \\ 2\delta c(e) & \text{otherwise.} \end{cases} \tag{3.1}$$

We impose the following restrictions on a *walk with partial edges* $P = ((\delta_1, e_1), \cdots, (\delta_\ell, e_\ell))$:

- The subsequence of edges with $\delta = 1$ has to be a walk.
- If $\delta_i < 1$, then $e_i$ has to be incident to the walk.
- If $\delta_i < 1$, then $e_i$ may not occur a second time in the walk.

While the first two conditions simply capture the concept of a *walk with partial edges*, the purpose of the third condition is to simplify the notation and analysis. For the CEOP the third condition is not a limitation: if a partial edge were to be included several times, it is easy to see that we could reduce the size $\delta$ of one partial edge and increase the size of the other, see Figure 3.2a. If the sum of the sizes is greater than 1, e.g., because we included an edge as complete edge ($\delta = 1$) and as a partial edge, then there would be no gain in profit from the latter partial edge.

For a walk with partial edges we define the sets $S_c = \{e_i \in \{e_1, \cdots, e_\ell\} \mid \delta_i = 1\}$ and $S_p = \{e_i \in \{e_1, \cdots, e_\ell\} \mid \delta_i < 1\}$. The objective of the CEOP now becomes to find a walk with partial edges $P = ((\delta_1, e_1), \cdots, (\delta_\ell, e_\ell))$ such that

$$\sum_{i=1}^{\ell} c'(\delta_i, e_i) \leq B, \text{ and}$$

$$\sum_{e \in S_c} \pi(e) + \sum_{e_i \in S_p} \delta_i \cdot \pi(e_i) \text{ is maximized.}$$

Using these definitions, we can show the following property.

**Lemma 20.** *Given a solution $P$ for instance $(G = (V, E), s, \pi, c, B)$ for the CEOP, there exists a solution $P'$, computable in $O(|P| \log |P|)$, with at most one partial edge, $c(P') \leq c(P)$, and $\pi(P') \geq \pi(P)$.*

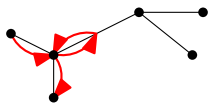


Figure 3.1: Example of a partial edge with $\delta = 0.5$, where the walk traverses half the edge and returns.

(a) (b)

Figure 3.2: Transforming two partial edges into one partial edge

*Proof.* Assume that $P$ has more than one partial edge. We can transform $P$ into $P'$ so that $P'$ has only one partial edge without sacrificing profit or increasing the cost of the walk. First, sort all partial edges of $P$ by their profit per cost ratio. Then, take two partial edges $(\delta, e), (\delta', e') \in P$ and let, w.l.o.g., $\pi(e)/c(e) \geq \pi(e')/c(e')$.

If we now increase $\delta$ and decrease $\delta'$ so that the total cost remains the same, then the total profit will not decrease. We do so until either $\delta = 1$ or $\delta' = 0$ (Figure 3.2). In the case $\delta = 1$, the partial edge $(\delta, e)$ is replaced by the complete edge $(1, e)$ twice. If $\delta' = 0$, the partial edge $(\delta', e')$ is removed. In both cases, the new solution has fewer partial edges, and by iterating this procedure, we obtain a solution $P'$ with at most one partial edge without increasing the budget and decreasing the profit. □

Using the previous lemma, we can show the following:

**Lemma 21.** *An $\alpha$-approximation for the CEOP implies a $2\alpha$-approximation for the EOP.*

*Proof.* Consider an instance $(G = (V, E), s, \pi, c, B)$ for the EOP, then we transform this instance into a CEOP instance. Let OPT and OPT′ be the values of optimal walks in the EOP and CEOP. Let $P'$ be the path we obtain by running the $\alpha$-approximation on the CEOP instance, giving us $\frac{\text{OPT}'}{\alpha}$ profit. W.l.o.g. assume that by Lemma 20, only one partial edge $(\delta, e) \in P'$ exists where $\delta < 1$. Then we create the following two walks for the EOP instance:

- Let $P_1$ be the walk that follows $P'$, but we do not traverse edge $e$.
- Let $P_2$ be the walk obtained by concatenating the shortest path from $s$ to one endpoint of edge $e$, then the edge $e$ itself, and finally the shortest path from the other endpoint back to $s$. Note that $P_2$ can traverse edge $e$ twice, namely if $e$ lies on one of the shortest paths.

If $\pi(e) < \frac{\text{OPT}'}{2\alpha}$, then $P_1$ collects at least $\frac{\text{OPT}'}{2\alpha}$ profit and obviously fits the budget. If $\pi(e) \geq \frac{\text{OPT}'}{2\alpha}$, then $P_2$ collects at least $\frac{\text{OPT}'}{2\alpha}$ profit. Also, $P_2$ fits the budget by preprocessing, since otherwise we would have removed $e$ from $G$. Since the path that collects OPT also exists in the CEOP instance, OPT′ ≥ OPT. Hence, choosing the walk $P_1$ or $P_2$ that collects the most profit gives us a path that gives us at least $\frac{\text{OPT}}{2\alpha}$ profit for the original EOP instance. □

Next, we need to approximate the CEOP.

**Theorem 22.** *An $\alpha$-approximation for the Orienteering Problem on a graph with profits 0 and 1 implies an $(\alpha+\varepsilon)$-approximation for the CEOP on graphs with polynomially bounded integer profits.*

*Proof.* For our reduction to work, we first scale profit function $\pi$ of our CEOP instance with $\frac{n^2}{\varepsilon}$, i.e. $\pi^*(e) = \frac{n^2}{\varepsilon}\pi(e)$ for each $e \in E$. Since each edge is scaled with the same scaling

factor, the problem remains equivalent; however, this scaling allows us to later bound the error introduced at each edge when reducing to the Orienteering Problem.

We then construct a new instance $((V', E'), s, \pi', c', B)$ for the Orienteering Problem. We assign each vertex a profit of either 0 or 1 and distribute the profit of an edge uniformly over the newly introduced vertices. For each vertex $v \in V$ we add a vertex $v$ to $V'$ with profit 0. Next, for each edge $e = \{v, w\} \in E$, we add vertices $u_i^e$ with profit 1 to $V'$, for $1 \le i \le \lceil \pi^*(e) \rceil$. We connect these vertices with edges $\{v, u_1^e\}$, $\{u_i^e, u_{i+1}^e\}$ for $1 \le i \le \lceil \pi^*(e) - 1 \rceil$, and $\{u^e_{\lceil \pi^*(e) \rceil}, w\}$ which are added to $E'$ each with cost $c(e)/\lceil \pi^*(e) + 1 \rceil$. Thereafter we run the $\alpha$-approximation algorithm on this new instance with the same starting point $s$ and budget $B$ to obtain walk $P'$. Furthermore, a transformation analogous to the transformation described with Figure 3.2 is applied on $P'$. Hence, we can assume, w.l.o.g., that for every edge $e \in E$, the corresponding chain of vertices is traversed only once if traversed partially.

Next, we iteratively build a walk with partial edges $P$ that work for our CEOP instance based on $P'$. First, let $v = s$, then we follow $P'$ until we reach a vertex $w$ that is part of our CEOP instance, i.e. $w \in V$.

- if $v \neq w$, then we simply add $(1, \{v, w\})$ to $P$.
- if $v = w$, then let $i$ be the number of distinct vertices traversed in $P'$ between $v$ and $w$. We then add $(i / \lceil \pi^*(e) + 1 \rceil, \{v, u\})$ to $P$, where $u \in V$ is the other endpoint of edge corresponding to the chain which is visited between $v$ and $w$ (when $i = 0$ no partial edge is added to $P$ and this step is skipped).

Then, we set $v := w$ and repeat the aforementioned steps until $P'$ is completely processed.

When we compare $P$ with $P'$ a small amount of profit may be lost. By construction, for each original edge $e$ we introduced $\lceil \pi^*(e) \rceil$ unit-profit vertices while the original edge had $\pi^*(e)$ profit, so the rounding in the reduction can increase the profit per edge by at most $\lceil \pi^*(e) \rceil - \pi^*(e) \le 1$ profit. Since the profit of a vertex can be only counted once, at this step at most $n^2$ additional profit is introduced.

Secondly, for edges where $v = w$, walk $P'$ collects $i$ profit by paying only a cost of $\frac{i \cdot 2c(e)}{\lceil \pi^*(e) \rceil + 1}$. Therefore, the corresponding partial edge added to $P$ collects at most $\frac{i \cdot \pi^*(e)}{\lceil \pi^*(e) \rceil + 1}$ profit. Hence, for each partial edge, the additional profit lost is at most

$$i - \frac{i \cdot \pi^*(e)}{\lceil \pi^*(e) \rceil + 1} = i \cdot \frac{\lceil \pi^*(e) \rceil + 1 - \pi^*(e)}{\lceil \pi^*(e) \rceil + 1} \le \lceil \pi^*(e) \rceil \cdot \frac{2}{\lceil \pi^*(e) \rceil + 1} \le 2,$$

where the last inequality uses $i \le \lceil \pi^*(e) \rceil$. Since, as mentioned above, at most one partial edge is introduced for each original edge, at most $2n^2$ profit is lost at this step. Thus, by combining the loss of rounding with the loss attributed to partial edges, we get:

$$\pi^*(P) \ge \pi'(P') - 3n^2. \tag{3.2}$$

Furthermore, let OPT* and OPT′ be the maximum profits obtainable in the scaled CEOP and the Orienteering Problem instances, respectively. Note that OPT* may be fractional due to a partial edge contributing $\delta\pi^*(e)$, whereas OPT′ is integral. By Lemma 20, there is an optimal CEOP solution with at most one partial edge $(\delta, e)$. All fully traversed edges map to traversing their entire chains and thus would incur no loss. For the (unique) partial edge, let $k = \lceil \pi^*(e) \rceil$ and traverse $i = \lfloor \delta(k+1) \rfloor$ unit-profit vertices on the corresponding

chain and return. This costs at most $2\delta c(e)$ and collects profit $i \geq \delta(k+1) - 1 \geq \delta\pi^*(e) - 1$ (as $k + 1 \geq \pi^*(e)$), hence the mapping loses at most one unit of profit. Therefore,

$$\mathrm{OPT}' \geq \mathrm{OPT}^* - 1. \tag{3.3}$$

Moreover, we know that

$$\pi'(P') \geq \frac{\mathrm{OPT}'}{\alpha}. \tag{3.4}$$

Thus, combining Equations 3.2, 3.3, and 3.4 gives us:

$$\pi^*(P) \geq \pi'(P') - 3n^2 \geq \frac{\mathrm{OPT}'}{\alpha} - 3n^2 \geq \frac{\mathrm{OPT}^* - 1}{\alpha} - 3n^2. \tag{3.5}$$

Scaling this back to the original profits results in:

$$\pi(P) \geq \frac{\mathrm{OPT}}{\alpha} - \frac{\varepsilon}{\alpha n^2} - 3\varepsilon \tag{3.6}$$

If $n \geq 1$ and since $\alpha \geq 1$, we know that $\frac{\varepsilon}{\alpha n^2} \leq \varepsilon$ and thus

$$\pi(P) \geq \frac{\mathrm{OPT}}{\alpha} - 4\varepsilon. \tag{3.7}$$

This bound implies $\pi(P) \geq \frac{\mathrm{OPT}}{\alpha+\varepsilon}$ provided $\mathrm{OPT} \geq 4\alpha(\alpha+\varepsilon)$. Hence it remains to handle the case when $\mathrm{OPT} < 4\alpha(\alpha+\varepsilon)$.

Because profits are integers, we enumerate all sequences of pairwise distinct edges of size $0 \leq t \leq \lceil 4\alpha(\alpha+\varepsilon) \rceil$, where each edge has a nonzero profit. For each such a sequence $S$ of profitable edges we consider both possible parities (i.e., orientations) of traversal for each edge. Furthermore, by Lemma 20 we know there can exist an optimal path that has at most 1 partial edge. Hence, we additionally guess a strictly partial edge $e^* = \{u, w\} \in E \setminus S$ and guess endpoint $u \in \{u, w\}$ from which the partial traversal of $e^*$ starts (and to which it returns).

A candidate walk $P$ is then constructed by starting from $s$, connecting consecutive edges of $S$ by shortest paths in $G$, and traversing each selected edge according to its chosen parity; finally the walk is routed back to $s$ via a shortest path. To incorporate the partial edge, we treat $u$ as an additional waypoint inserted between $e_j$ and $e_{j+1}$: we connect the exit endpoint of $e_j$ to $u$ by a shortest path, then traverse a $\delta$-fraction of $e^*$ and return to $u$, and then connect $u$ to the entry endpoint of $e_{j+1}$ by a shortest path. Again, we enumerate all possible values of $0 \leq j \leq t$.

Let $c(P_{\text{base}})$ be the cost of the resulting walk where the partial traversal of $e^*$ is omitted (i.e., with $\delta = 0$ but still including the two shortest paths to and from $u$). Then, let $S = B - c(P_{\text{base}})$ be the remaining budget and if $S < 0$, the candidate is infeasible. Moreover, if $S \geq 2c(e^*)$, then $e^*$ can be traversed completely as a detour (forth and back) and this case is handled by an enumeration in which $e^*$ is traversed as a complete edge. Otherwise, we set

$$\delta = \frac{S}{2c(e^*)},$$

which satisfies $0 \leq \delta < 1$ and maximizes the additional collected profit for this candidate. We then check every candidate, selecting the one that yields the highest profit while staying within the budget.

Then, if the most profitable walk with total cost at most $B$ among these enumerated

walks collects at most $4\alpha(\alpha+\varepsilon)$ profit, this walk collects OPT profit and can be reported. Since $4\alpha(\alpha+\varepsilon)$ is a constant for fixed $\alpha$ and $\varepsilon$, the enumeration runs in polynomial time. Otherwise, if the most profitable walk within budget $B$ collects more than $4\alpha(\alpha+\varepsilon)$ profit, the aforementioned reduction is applied. Thus, in both cases, the returned walk achieves an $(\alpha+\varepsilon)$-approximation for the CEOP. □

Finally, we need to reduce the Orienteering Problem on graphs with profits 0 and 1 to the Orienteering Problem on graphs with unit profit.

**Lemma 23.** *An $\alpha$-approximation for the Orienteering Problem on graphs with unit profits implies an $(\alpha+\varepsilon)$-approximation for the Orienteering Problem on graphs with profits 0 and 1.*

*Proof.* Let $((V,E),s,\pi,c,B)$ be the Orienteering Problem instance where $\pi: V \to \{0,1\}$. We construct a new instance $((V',E'),s,\pi',\hat{c},B)$ for the Orienteering Problem where all vertices will have unit profit (i.e. for all $v' \in V'$ we have $\pi'(v')=1$). To build this instance:

- Add all vertices $v \in V$ to $V'$ and all edges $e \in E$ to $E'$.
- For each vertex $v$, introduce $n^2 \cdot \pi(v)$ extra vertices with unit profit and connect them to $v$ with edges of cost 0. These extra vertices ensure that profits from the original graph are preserved while converting to unit profits.

On this newly constructed instance with unit profit for all vertices, we apply the $\alpha$-approximation algorithm to obtain a walk $P'$. From this walk $P'$, we construct a walk $P$ for the original instance by following $P'$ but excluding any edge that is incident to a newly introduced vertex. This ensures that the resulting walk $P$ only traverses original edges from $E$ and visits original vertices from $V$.

Let OPT and OPT′ denote the optimal profits of walks in the original and modified instances, respectively. Since every vertex $v \in V$ with $\pi(v)=1$ corresponds to a group of size $(n^2+1)$ vertices in the modified instance (and if $\pi(v)=0$, there is still exactly one vertex in this group), we know that a walk that visits $g$ distinct original vertices in $P$ corresponds to at most $g \cdot (n^2+1)$ unit-profit vertices in $P'$. Hence, for walk $P'$ and its corresponding $P$ we obtain

$$n^2 \cdot \pi(P) \geq \pi'(P') - n.$$

Moreover, the $\alpha$-approximation guarantees that $\pi'(P') \geq \frac{\mathrm{OPT}'}{\alpha}$. By construction of the instance we also have $\mathrm{OPT}' \geq n^2\mathrm{OPT}$, as any optimal walk in the original instance visiting OPT profitable vertices can be extended in the modified instance by collecting $n^2$ additional vertices for each profitable original vertex at zero additional cost.

Combining these inequalities we obtain

$$n^2 \cdot \pi(P) \;\geq\; \pi'(P') - n \;\geq\; \frac{\mathrm{OPT}'}{\alpha} - n \;\geq\; \frac{n^2\mathrm{OPT}}{\alpha} - n. \tag{3.8}$$

Thus, solution $P$ in the original instance has profit

$$\pi(P) \geq \left(\frac{1}{\alpha} - \frac{1}{n}\right)\mathrm{OPT}, \quad \text{when } \mathrm{OPT} \geq 1.$$

Whether $\mathrm{OPT} \geq 1$ can be easily checked by seeing if there exists a shortest path between $s$ and a vertex $v \in V$ with $\pi(v)=1$, such that this shortest path is at most half the budget $B$. Finally, we must handle instances where $(1/\alpha - 1/n)$ is smaller than $1/(\alpha+\varepsilon)$. This

inequality only occurs when $n < (\alpha + \varepsilon)/\varepsilon$, which is a constant threshold depending on fixed constants $\alpha$ and $\varepsilon$. Thus, when $n$ is sufficiently small, we solve the instance by brute force. We enumerate all sequences $(v_1, \cdots, v_t)$ of pairwise distinct vertices from $V$ for all lengths $0 \leq t \leq n$. For each sequence we construct a candidate walk by starting at $s$ and connecting consecutive vertices with shortest paths in $G$ and finally routing back to $s$. We then compute the total cost of the resulting walk and keep the most profitable candidate walk whose total cost is at most $B$. Since $n$ is bounded by a constant in this case, this enumeration runs in polynomial time and returns an optimal solution for the instance whenever $n < (\alpha + \varepsilon)/(\varepsilon)$ Otherwise, when $n$ is larger we can use the aforementioned reduction to obtain an $(\alpha + \varepsilon)$-approximation. □

Finally, with Lemmas 19, 20, 23, Theorem 22, and the $(2 + \varepsilon)$-approximation for the Orienteering Problem from [21], we obtain our main result.

**Corollary 24.** *There exists a $(4 + \varepsilon)$-approximation for the EOP.*

## 3.2 Touring in Practice

In this section, we study the *Touring Problem* in a practical setting on real-world road networks. The Touring Problem generalizes the *Edge Orienteering Problem*, which seeks a walk of bounded length that maximizes the total collected edge profit. Verbeeck, Vansteenwegen, and Aghezzaf [92] propose a branch-and-cut algorithm for solving the AOP exactly and an iterative local search meta-heuristic for larger instances. Lu and Shahabi. [59] improve this meta-heuristic using spatial database techniques.

Touring is also closely related to minimum-cost path computation: one can reduce the cost of desirable edges and compute shortest paths. In these *efficient* paths desirable edges are preferred when given the option. However, for round-trip generation this approach is problematic, as the trivial tour of length 0 becomes optimal. A common workaround introduces intermediate waypoints and concatenates shortest paths between consecutive waypoints to form a tour. This approach has been used for jogging [38] and cycling [76] route planning. The touring objective captures the round-trip setting more directly by combining edge profits with an explicit global tour-quality term.

**Problem definition.** An instance of the Touring Problem is given by $(G = (V, E), s, \pi, c, B, q, w)$, where $G$ is an undirected graph with vertex set $V$ and edge set $E$, $s \in V$ is the starting vertex, $\pi : E \to \mathbb{R}_{\geq 0}$ assigns a profit to each edge, $c : E \to \mathbb{R}_{\geq 0}$ assigns a cost (e.g., distance or travel time), and $B \in \mathbb{R}_{\geq 0}$ bounds the total cost. The function $Q$ is a global quality measure on tours and $w \in [0, 1]$ is the weight of this quality term.

A tour $\mathcal{T}$ corresponds to a closed walk in $G$ that starts and ends at vertex $s$. The total cost $c(\mathcal{T})$ is the sum of the costs of the edges traversed by $\mathcal{T}$. The total profit $\Pi(\mathcal{T})$ is the sum of the profits of edges contained in $\mathcal{T}$, where each edge contributes at most once even if it is traversed multiple times.

Given a budget $B$, start vertex $s$, and quality weight $w$, the objective of the Touring Problem is to find a tour $\mathcal{T}$ with $c(\mathcal{T}) \leq B$ that maximizes the total profit:

$$\Pi(T) = (1 - w) \cdot \Pi(\mathcal{T}) + w \cdot Q(\mathcal{T}).$$

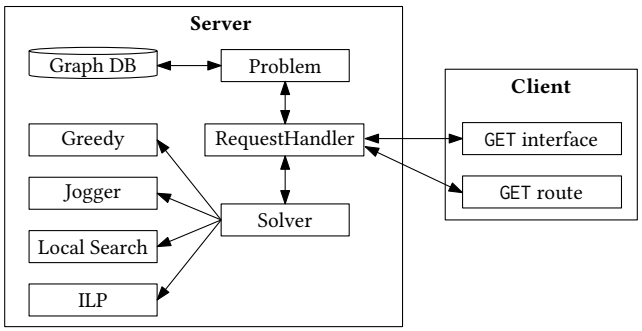



Figure 3.3: System architecture.

**Global quality measure.** As an example of a global quality function $Q(\mathcal{T})$, we use the polygonal area induced by the tour geometry. We represent $\mathcal{T}$ as a closed polygonal chain and compute its area using the shoelace formula. For simple (non-self-intersecting) tours this equals the enclosed area and favors "round" routes over long thin strips. However, for self-intersecting tours, the shoelace formula computes the *algebraic* area, where regions enclosed with opposite orientation cancel. This is not a flaw of the computation, but a choice of quality signal. The shoelace formula rewards tours that enclose area in a consistent direction and discourages tours that repeatedly retrace or cross themselves. This choice is especially attractive in our setting because this calculation is computationally cheap and supports efficient recomputation after local modifications, which we exploit in Section 3.2.2.

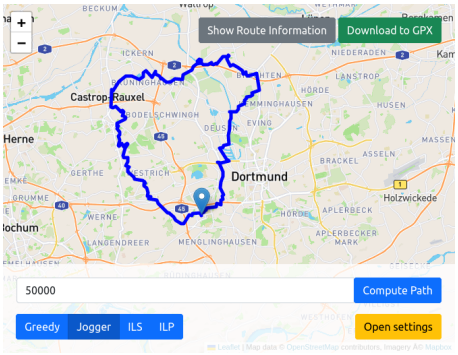
### 3.2.1 Experimental setup (Tour4Me)

To study the touring objective on real street networks, we implemented a prototype system, Tour4Me, that instantiates touring instances from road-network data and user-style preferences, that can run different solvers under identical inputs, and visualizes and summarizes the resulting tours. The purpose of Tour4Me is not to present a general-purpose framework, but to make the touring objective operational and enable a practical comparison of algorithmic approaches on realistic instances. Additionally, Tour4Me can be used as a testbed for testing new algorithms and heuristics for the Touring Problem, and to explore the trade-offs between solution quality and runtime in practice. This is particularly relevant for educational purposes, as it allows students to experiment with different algorithms and see their performance on real-world data. The source code for Tour4Me is available on GitHub[2].

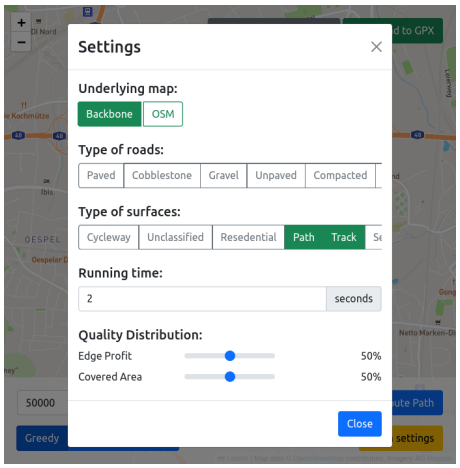
**Implementation overview.** Tour4Me is implemented as a one-page web application with a C++ back-end to support efficient route computation (Figure 3.3). A REST API is hosted using *libhttpserver*[3] and serves both the web interface and route computation requests. Route computations are triggered via GET requests and handled by a request handler that instantiates the problem using the selected data set and user preferences, runs the chosen solver, and returns the computed tour to the front-end for visualization.

The aforementioned web application is mostly for the purpose of demonstrating the different algorithms on real-world data. To run actual experiments one can alternatively

[2] https://github.com/traMectory/tour4me
[3] https://github.com/etr/libhttpserver




(a) Map view with a calculated route.




(b) Settings panel controlling the user's preferences.

Figure 3.4: The user interface of TOUR4ME.

run them directly on the C++ back-end codebase to avoid the HTML overhead.

**Data.** For our experiments we store a sequence of overlapping map tiles representing the street network as separate files on the server. When a user requests a tour, the system selects the tile that best matches the chosen starting point and loads the corresponding graph into memory.

The tiles are downloaded from *OpenStreetMap*[4] (OSM) using *OSMnx* [15]. OSMnx downloads and topologically simplifies the raw OSM data to reduce graph size. The resulting graphs are stored using a modified version of the DIMACS naming convention[5].

Even after simplification, full road graphs still contain on the order of $10^5$ nodes, which is too large for exact methods in our setting. We therefore also provide a reduced representation by selecting a set of "interesting" edges and simplifying the induced subgraph. We denote this simplified subgraph as the *backbone*. The backbone is intended to preserve salient structure while substantially reducing size, and it allows us to study the trade-off between runtime and solution quality.

**Interface.** The user interface for demonstration purposes (Figure 3.4) consists of a background map on which a starting point can be selected and a target distance (budget) can be specified. Additional controls are available in the settings panel. Users can choose between the full street network and the reduced backbone.

We encode user-style preferences by assigning profit values to edges based on road types (OSM tag `highway`) and surface types (OSM tag `surface`). For each tag, edges can be assigned zero, positive, or negative profit. Additionally, a runtime limit can be set, which is particularly relevant for the more expensive algorithms described in Section 3.2.2. Finally, we select the relative importance of edge profit versus covered area via the weight $w$.

Once parameters are chosen, the front-end sends a `GET` request containing the start

[4] `https://www.openstreetmap.org/`
[5] `http://lcs.ios.ac.cn/~caisw/Resource/about_DIMACS_graph_format.txt`

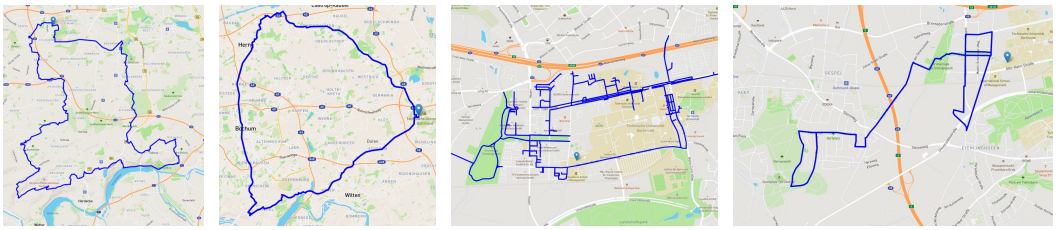


Figure 3.5: Examples of tours computed with different algorithms. From left to right: minimum cost tour with waypoints, iterated local search based on minimum cost tour and for area, greedy selection, and integer linear program.

location, budget, and preferences. The server computes a tour with the selected solver and returns a JSON[6] response describing the route. The route is visualized on the map and summary statistics (e.g., length, profit, quality score) are displayed. The final route can be exported as a GPX file for use on navigation devices.

### 3.2.2 Algorithms

We include five different representative algorithms in our case study. The greedy solver Greedy Selection quickly constructs a tour by repeatedly selecting high-profit edges. The waypoint-based solver MinCost builds a cycle from three efficient paths. Next, we implement a meta-heuristic Iterative Local Search that starts from a waypoint-based tour and improves it via iterative local modifications. Finally, we propose an integer linear programming model Integer Linear Programming for the Touring Problem and solve it using an MILP solver.

#### Greedy Selection

The greedy approach Greedy Selection computes a cycle starting at vertex $s$ by iteratively picking edges with highest profit. To ensure that the partial tour can still be closed, Greedy Selection only selects an edge whose endpoint can reach $s$ within the remaining budget (via a shortest path with respect to cost). We refer to such edges as *valid candidates*.

The algorithm begins by selecting the highest-profit valid candidate edge incident to $s$. It then updates the remaining budget and repeats the selection from the new endpoint until it returns to $s$ or no valid candidate exists. If the process terminates at a vertex $v \neq s$, the algorithm adds a shortest path from $v$ back to $s$ to close the tour.

This approach is extremely fast but typically yields tours that are far from optimal and may have poor shape or repeated segments. For example, in our experiments, computing a route of length 100 km takes on average 24 ms, but the resulting tour often lacks desirable global structure (Figure 3.5).

[6] https://www.json.org/

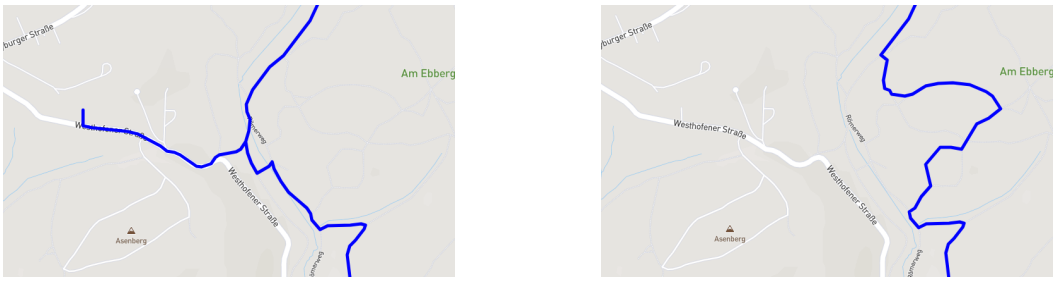



Figure 3.6: Example of part of a tour before (left) and after (right) applying ILS.

### MinCost with Waypoints

The second algorithm is inspired by the waypoint stitching approach of Gemsa, Pajor, Wagner, and Zündorf [38]. In contrast to GREEDY SELECTION, it explicitly trades off profit and cost by favoring efficient edges, i.e., edges with high profit relative to their cost. We define an edge inefficiency measure $\gamma : E \to \mathbb{R}_+$ as

$$\gamma(e) = \frac{c(e)}{10^{q(e)-1} + 10^{3q(e)-4}},$$

Where $q(e) = c(e)/\pi(e)$ represents the quality of an edge, i.e. the desirability of the edge. In the case where $\pi(e) = 0$, we set $q(e) = \infty$ and thus $\gamma(e) = \infty$. Shortest paths with respect to $\gamma$ prefer edges that provide high profit per unit cost.

The algorithm aims to construct a tour resembling an equilateral triangle composed of three efficient paths of length approximately $B/3$, with $s$ as one of the triangle vertices. First, it computes a ring $R_s$ of candidate vertices that can be reached from $s$ by a shortest path in the $\gamma$-weighted graph such that the cost of this path in the original graph lies in

$$\left[\frac{B}{3} - \frac{B}{10}, \frac{B}{3} + \frac{B}{10}\right].$$

For each $r \in R_s$, it constructs an analogous ring $R_r$. For each vertex in $R_s \cap R_r$, the algorithm forms a candidate tour by concatenating shortest paths in the $\gamma$-weighted graph from $s$ to $r$, from $r$ to the intersection vertex, and back to $s$, and evaluates the tour in the original objective. After limiting the size of $R_s$ to control runtime, this approach can compute tours of length 100 km in approximately 0.5 s in our setup.

### Iterative Local Search

The ITERATIVE LOCAL SEARCH solver adapts the iterative local search method proposed for the AOP by Verbeeck et al. [92]. It starts from an initial solution computed by MINCOST and attempts to improve it by repeatedly replacing short subpaths.

In each iteration, the current tour $S$ is modified by removing a subpath $P$ from a vertex $u$ to a vertex $v$. The removed subpath is determined by two parameters: a position $p$ on the tour where removal starts and a length parameter $l$ indicating how many edges are removed. Both parameters are initialized to 1 and increased by one in each iteration. The position parameter $p$ wraps around when it reaches $s$, and $l$ wraps around when it reaches the length of the tour.

After removing $P$, ITERATIVE LOCAL SEARCH searches for an alternative path $P'$ connecting

$u$ and $v$ that respects the remaining budget, i.e.,

$$c(P') \leq B - c(S \setminus P),$$

and yields higher objective value. The local move is implemented as a depth-first search with a maximum search depth $D$; in the demonstration we use $D = 10$. As soon as a feasible path reaching $v$ is found, the algorithm evaluates whether replacing $P$ with $P'$ increases the objective. If so, the tour is updated. This process continues until the time limit is reached.

In practice, Iterative Local Search is effective at eliminating small inefficiencies (e.g., reducing duplicated segments) and making local shape improvements, as illustrated in Figure 3.6. In our experiments, noticeable improvements are typically achieved within a few seconds.

### Integer Linear Programming

Finally, we introduce an exact solver Integer Linear Programming based on an integer linear programming formulation of the Touring Problem, solved using Gurobi. We encode an instance $\mathcal{I} = (G, w, \pi, B, v_0)$ as an Integer Linear Programming model, where $v_0$ is the start vertex.

**Canonical cycles.** A feasible solution cycle contains at most $L := \frac{B}{c_{\min}}$ edges, where $c_{\min}$ is the minimum edge cost. In our demonstration we fix $L = 30$ to limit model size.[7] Any feasible cycle $P = (v_0, \dots, v_i, \dots, v_0)$ of length at most $L$ can be extended to length exactly $L$ by appending extra occurrences of $v_0$ at the end without changing the cost or profit. We refer to such a length-$L$ representation as a *canonical cycle*.

**Variables.** For each (undirected) edge $v_i v_j \in E$, we introduce: a binary variable $b_{ij}$ indicating whether the edge is used at least once, an integer variable $n_{ij}$ indicating how many times it is traversed, and binary variables $p_{kij}$ indicating whether $v_i v_j$ is the $k$-th edge of the canonical cycle, for $k \in \{1, \dots, L\}$. For vertices, we introduce binary variables $p_{qi}$ indicating whether $v_i$ is the $q$-th vertex of the canonical cycle, for $q \in \{1, \dots, L+1\}$.

**Objective.** Ignoring the area term, the profit objective is $\max \sum_{v_i v_j \in E} \pi(v_i v_j) \cdot b_{ij}$. To incorporate area maximization via the shoelace method, we add $\sum_{e_{ij}} \sum_{k=1}^{L} \left( p_{kij} \cdot fA_{ij} + p_{kji} \cdot bA_{ij} \right)$, where $fA_{ij}$ and $bA_{ij}$ are the signed area contributions of traversing $e_{ij}$ forward and backward, respectively.

**Constraints.** We introduce the following seven constraints in our formulation which model the Touring Problem.

$$\forall v_i v_j \in E, \sum_{k \in [1, \cdots, L]} p_{kij} = n_{ij} \tag{3.9}$$

$$\forall v_i v_j \in E, b_{ij} \leq n_{ij} \tag{3.10}$$

[7] This bound is chosen for computational reasons in the demonstration setup.

$$\forall k \in [1, \cdots, L], \sum_{v_i v_j \in E} p_{kij} = 1 \tag{3.11}$$

$$\forall q \in [1, \cdots, L+1], \sum_{v_i \in V} p_{qi} = 1 \tag{3.12}$$

$$\forall v_i v_j \in E, p_{ki} + p_{(k+1)j} \geq 2 \cdot p_{kij} \tag{3.13}$$

$$p_{10} = 1, p_{(L+1)0} = 1 \tag{3.14}$$

$$\forall v_i v_j \in E, \sum_{v_i v_j \in E} c(v_i, v_j) \cdot n_{ij} \leq B \tag{3.15}$$

Note that this does not match our profit function exactly, as we do not give a penalty for when the tour is deviating from the target distance as the original profit function is not linear. However, since the profit function is monotone in the cost, we can expect that the optimal solution of this ILP will be close to the target distance.

### Waypoint Solver

The WAYPOINT algorithm presented in this section has been implemented as part of Kamberg's Master's thesis under my supervision [50]. This algorithm aims to have a lower runtime than the aforementioned algorithms, whilst being more robust and providing solutions with higher quality. As the name suggests, the WAYPOINT solver guesses waypoints and connects these with the same efficient paths as used by the MINCOST algorithm (Section 3.2.2).

To determine the waypoints, multiple circles are projected onto the street network, each of these circles corresponds to a candidate tour. For each circle we calculate the *ideal* waypoints at regular intervals. Since there is usually no waypoint located exactly at an ideal location, we search in the street network for a vertex of the street network that is closest to the coordinates of the ideal waypoint. Furthermore, the number of waypoints depends on the desired length of the round trip, as a larger circle offers more space for waypoints. As with the MINCOST algorithm, these waypoints are connected with efficient paths and the quality of the full tour is calculated. Finally, the candidate tour with the highest quality is selected and returned as the best route.

Furthermore, the radius of the projected circle is based on the desired budget. However, since there is rarely a perfect arc between two points in a street network, the efficient routes are often longer. Therefore, the radius must be adjusted such that the lengths of the candidate routes match with the desired budget. To this end, the variable RADIUS CORRECTOR is introduced that decreases the radius of the circle by a certain factor. This corrector variable is set experimentally.

This Waypoint solver also includes a post-processing step to create more natural routes. When concatenating two efficient paths between consecutive waypoints, it may occur that a short segment of the network is traversed twice: first when approaching a waypoint and immediately afterwards when leaving it. Such situations result in a small "there-and-back" detour around the waypoint. Since these detours are typically undesirable for round trips, they are detected and removed. For each waypoint, we check whether the path segment leading to the waypoint overlaps with the segment leaving it. If so, the overlapping section is identified by iteratively comparing predecessor and successor nodes around the waypoint. The corresponding nodes of this redundant segment are then removed while keeping the tour connected, resulting in a shorter and more natural route.

### 3.2.3 Reduced Graphs

A key ingredient for keeping the runtime of the different solvers low is the use of a *backbone*, i.e., a reduced version of the full OSM graph. Since shortest-path computations (in our implementation based on A* search) scale strongly with graph size, reducing the graph can speed up tour construction substantially. The challenge is to reduce aggressively while still preserving enough structure so that high-quality tours remain feasible.

A second complication is that the touring instances are highly customizable: profits depend on user-selected preferences over road and surface types. As a consequence, there is no single reduced graph that is optimal for all profiles. Ideally, different profiles would come with different backbones. Of course, the number of possible preference combinations is far too large to precompute and store a backbone for each. What we would like instead is a method that can construct a backbone *for a given preference profile* (or for a small representative set of profiles), so that we can study the trade-off between runtime and solution quality under realistic customization.

In the following, we consider several backbone constructions. We start with an existing backbone based on curated bicycle routes. Furthermore, we look at two new methods as implemented by Kamberg [50]. The first new backbone is derived from efficient paths with respect to the current profit/cost trade-off, and we look at contraction hierarchies to improve shortest path calculations.

#### Backbone from Bicycle Routes

The first backbone we introduce is the *Backbone from Bicycling Routes* (BBR). The BBR consists of roads that are part of published bicycle routes. Since these routes are curated and maintained by local or regional organizations, we treat them as reasonable for cycling. In OpenStreetMap, such routes can often be identified via relations tagged with `route=bicycle`. Figure 4.1 shows the resulting BBR for the area around Dortmund.

While the BBR yields a useful reduced graph in settings where such route data is available, it is not a general solution. First, curated bicycle-route relations are not consistently available worldwide, so coverage can be sparse or missing depending on the region. Second, the approach is difficult to adapt to different preference profiles. Even within cycling, requirements vary substantially (e.g., road bike vs. mountain bike), and for other activities such as jogging or hiking the relevant notion of “good” infrastructure differs again. Since we cannot rely on having curated route maps for every region and every set of preferences, we also need a backbone construction that can be derived more directly from the underlying street network and the chosen profit model.

#### Backbone from Efficient Paths

The second backbone is implemented by Kamberg [50]. Here we derive a reduced graph from efficient paths under the same profit–cost trade-off used by the solvers. The intuition is that edges that repeatedly appear on efficient connections are plausible building blocks for round trips, whereas edges that are rarely used by efficient paths are less relevant and can be removed.

The construction starts by selecting the four extreme vertices of the map (northwesternmost, northeasternmost, southeasternmost, and southwesternmost). We connect neighboring corner vertices by efficient paths, producing four boundary paths. On each

boundary path we then choose a fixed number of intermediate vertices (20, including endpoints), evenly spaced with respect to vertex order along the path. To increase coverage, we connect these sampled vertices across different boundary paths: each sampled vertex on one boundary path is connected via efficient paths to all sampled vertices on the other three boundary paths. The union of the edges contained in these paths forms the initial backbone candidate.

A known issue of using only global boundary-to-boundary paths is that many of them share the same central segments, which reduces path diversity in the middle of the map. Since short round trips often lie in the interior, we address this by additionally subdividing the map into four quadrants and applying the same construction within each quadrant. This yields efficient paths at multiple spatial scales and produces a denser set of retained edges in central regions.

Overall, this method produces a backbone that is directly derived from the underlying network and the current preference profile, without relying on curated route relations, while still preserving the graph structure that is most likely to matter for round-trip generation.

#### Contraction Hierarchy

In addition to reducing graph sizes via backbones, another technique to speed up shortest path queries is to use *contraction hierarchies* (CH). CH orders vertices by "importance" and adds shortcut edges, such that search spaces become significantly smaller. Since Tour4Me currently uses undirected graphs, we implement an undirected CH variant, following the general approach of Geisberger, Sanders, Schultes, and Delling [37]. To align CH with the Waypoint-Solver, we restrict waypoint candidates to a fixed set of high-ranked vertices (e.g., the 10,000 most important nodes). This ensures that the solver's geometric waypoint guesses snap to structurally relevant locations, while also improving runtime since queries tend to reach higher-level vertices quickly.

We order the nodes based on their importance. Furthermore, we combine *edge difference* (shortcuts introduced minus incident edges) with the *number of deleted neighbors* (already contracted neighbors), weighted 0.75 and 0.25, respectively; the latter encourages a more even spatial distribution of important nodes, which is beneficial for waypoint placement. When a node is selected for contraction, a new edge is introduced between two incident vertices which has the combined cost and quality of the two original edges.

### 3.2.4 Evaluation

In this section we evaluate and compare the different solvers and backbones. First, we look at small instances to compare the Greedy Selection and Integer Linear Programming solvers. Next we look at larger instances where we look at how much Iterative Local Search improves routes generated by the MinCost solver. Finally we compare MinCost with Waypoint on different backbones.

#### Small Instances

Figures 3.7–3.9 compare Greedy Selection with the ILP solver under time limits of 60 s, 90 s, and 120 s (denoted ILP60/90/120). Recall that these limits only affect the ILP solver;

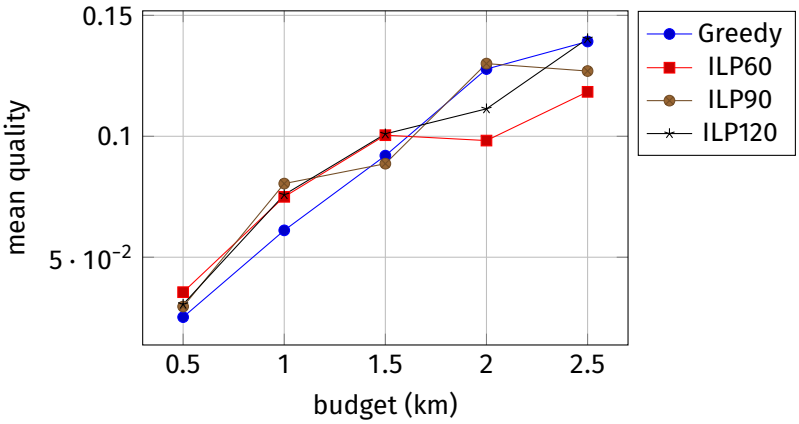



Figure 3.7: Mean quality for different solvers for small instances focused on profit.

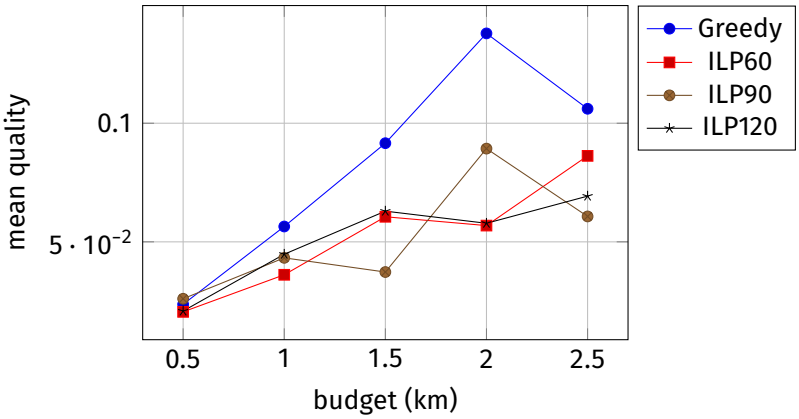



Figure 3.8: Mean quality for different solvers for small instances with a balanced focus on profit and area.

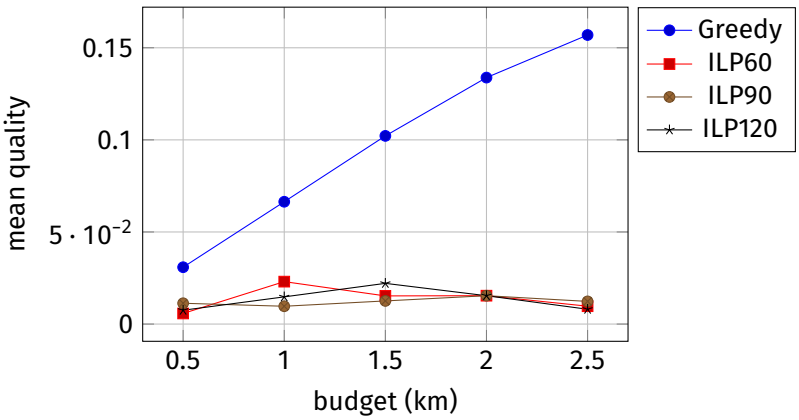



Figure 3.9: Mean quality for different solvers for small instances with a focus on area.

Greedy Selection does not use the limit parameter. Across all preference settings, increasing the budget yields higher mean quality for all solvers, with the strongest gains when moving from the smallest budgets (500m–1000m) to medium budgets (1500m–2000m). This is expected: larger budgets allow the tour to incorporate more profitable edges and/or form tours with better global shape.

**Profit-focused objective ($w$=0).** In Figure 3.7, ILP benefits from additional runtime, particularly at the larger budgets, where ILP90/ILP120 close most of the gap to Greedy Selection and occasionally match it. At the same time, Greedy Selection remains highly competitive and reaches comparable quality at a fraction of the computation time. Overall, the plot suggests diminishing returns from increasing the ILP time limit beyond 90 s: ILP120 improves over ILP60, but the improvement is modest relative to the additional runtime.

**Balanced objective ($w$=0.5).** Figure 3.8 shows that Greedy Selection consistently manages to achieve the highest mean quality for medium and large budgets. The ILP variants improve with budget as well, but the additional time (60→120 s) yields only small gains and does not close the gap to Greedy Selection. This indicates that, under a balanced profit/area objective, the ILP formulation and time limits used here tend to spend most of their effort finding feasible structures, while Greedy Selection already produces tours with good overall objective value.

**Area-focused objective ($w$=1).** In the area-only setting (Figure 3.9), Greedy Selection clearly dominates: its mean quality increases almost monotonically with budget, while the ILP curves remain close to zero. This suggests that with the current model size bound and time limits, the ILP approach struggles to exploit the geometric quality term, whereas Greedy Selection tends to produce tours that enclose larger, more consistently oriented regions as additional budget becomes available.

#### Iterative Local Search Improvements

To quantify how much Iterative Local Search improves an existing tour, we apply it as a post-processing step to tours generated by the waypoint-based solver MinCost and measure the *relative* change in objective value. For a fixed instance (start vertex, budget, and preference profile), recall that $\Pi(\cdot)$ is the touring objective. We define the relative improvement as

$$\mathrm{RI} = \frac{\Pi(\mathcal{T}_{\text{Iterative Local Search}}) - \Pi(\mathcal{T}_{\text{MinCost}})}{\Pi(\mathcal{T}_{\text{MinCost}})},$$

and report the mean relative improvement over all runs for each budget and Iterative Local Search time limit.

Figure 3.10 shows the resulting mean relative improvements for the balanced objective ($w$=0.5). Across all budgets, Iterative Local Search yields positive gains and improves the initial waypoint tour quickly: most of the improvement is obtained within the first few seconds, after which additional runtime produces diminishing returns. This matches the design of the search: early iterations tend to remove obvious local defects (short detours, repeated segments, and small shape inconsistencies), whereas later iterations spend more effort exploring alternatives that only rarely yield additional objective gain.

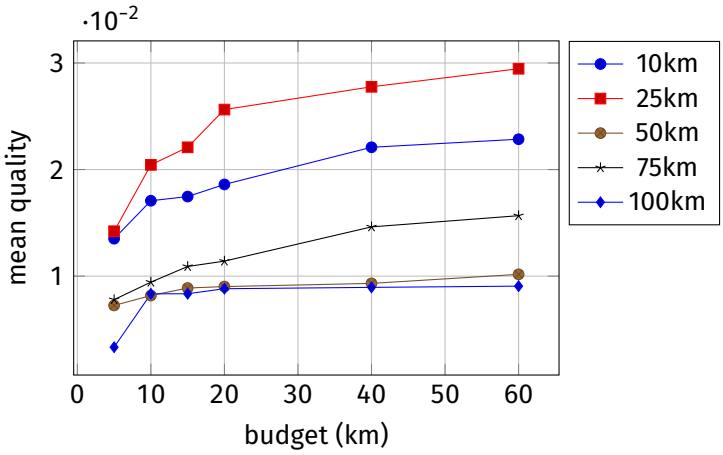



Figure 3.10: Mean quality improvements from applying ILS to routes generated by the MinCost solver.

| Budget | 10km | 25km | 50km | 75km | 100km |
|---|---|---|---|---|---|
| MinCost (OSM) | 99.7% | 98.3% | 82.7% | 73.0% | 80.0% |
| WP (OSM) | 99.0% | 100.0% | 100.0% | 100.0% | 100.0% |
| WP (OSM-nde) | 99.0% | 100.0% | 100.0% | 100.0% | 100.0% |
| WP (BBR) | 96.0% | 99.0% | 100.0% | 100.0% | 100.0% |
| WP (BE20) | 80.3% | 93.7% | 97.0% | 98.7% | 99.7% |
| WP (CH) | 98.3% | 99.7% | 100.0% | 100.0% | 100.0% |

Table 3.1: Percentages of successful queries.

Furthermore, the relative benefit of Iterative Local Search depends on the tour length. For shorter and medium budgets (10-25 km), the relative improvement is largest and continues to grow noticeably with time. For longer tours (50-100 km), the relative improvement is smaller and the curve flattens earlier. This is likely because longer tours have more complex global structure which is altered less by the local moves of Iterative Local Search, whereas shorter tours are more likely to contain local inefficiencies that have a higher impact.

### Waypoint and Backbones

**Robustness.** A major design goal of the Waypoint solver is improved robustness, i.e., returning a valid route reliably across different target budgets. Table 3.1 shows that the baseline approach (MinCost on OSM) succeeds in almost all cases for small budgets, but its success rate drops for larger budgets. In contrast, the Waypoint solver remains highly reliable across all budgets: using OSM or OSM-nde as backbone it achieves (almost) perfect success rates for all target budgets, and it also remains at 100% for budgets of 50 km and above when using BBR. With contraction hierarchies (WP (CH)), the success rate is near-perfect already for short budgets and reaches 100% from 50 km onwards.

The failures of the Waypoint solver with reduced backbones can be explained by sparsity effects at short budgets: for very small target lengths only a small number of waypoints is used, and on a sparse backbone multiple waypoint locations may collapse to the same node. In this case, subsequent route construction and post-processing (e.g., detour re-

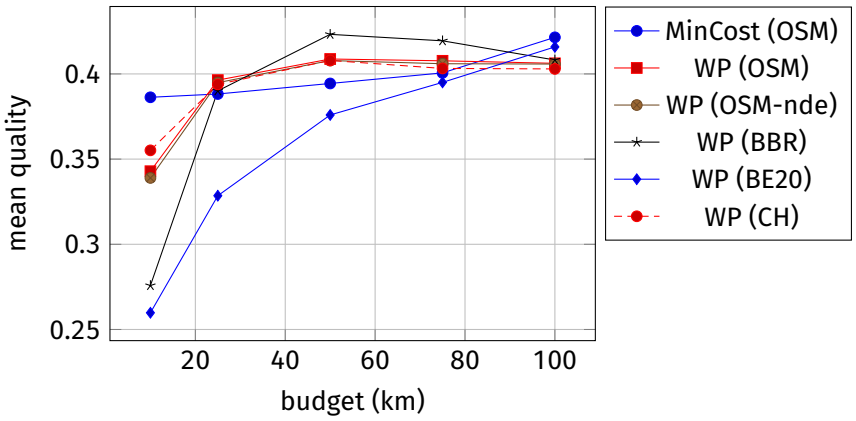



Figure 3.11: Mean quality for different solvers and budgets

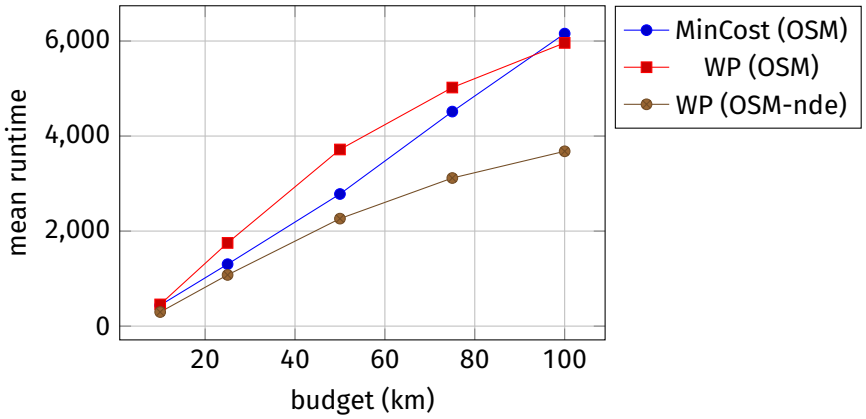



Figure 3.12: Runtimes for different solvers and target distances on OSM-maps

moval) can degenerate so that no meaningful route remains. Conversely, the baseline method fails more often for larger budgets because its sampling/search region grows with the target length while the number of retained candidate nodes is limited, reducing the effective node density and increasing the probability that no valid connecting structure is found.

**Quality.** Next, we evaluate the quality of MinCost and the Waypoint with different backbones. Figure 3.11 shows the mean quality for the different solvers and backbones. The MinCost solver performs throughout the range of budgets very consistently. However, apart from the lowest distances, the Waypoint solver outperforms the MinCost solver no matter the underlying backbone.

This fact can likely be attributed to the removal step of detours, since this step can heavily impact the length of a tour and thus the quality. Furthermore, this aligns with the lower quality of tours generated in the sparser graphs, as these graphs increase the chance that a path is taken twice.

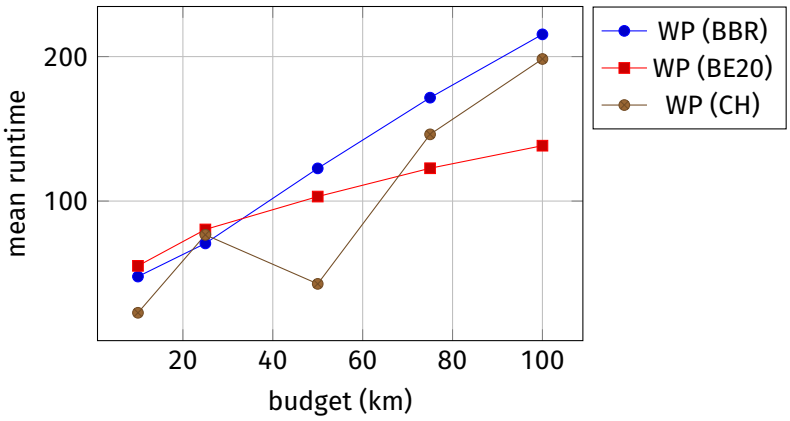



Figure 3.13: Runtimes for different solvers and target distances

**Runtime.** Figure 3.13 shows the difference in runtimes of MinCost and the Waypoint with different backbones. We see that the MinCost solver has the largest runtime and reducing the size of the street network significantly improves the runtime.

### Summary

Overall, the evaluation highlights a trade-off between runtime and solution quality across the different approaches. The ILP-based method provides useful reference points on small instances, whereas for larger real-world networks the waypoint-based approaches appear more effective in practice. In particular, the Waypoint solver achieves high robustness and good solution quality across the tested budgets, while reduced graph representations significantly improve runtime. Furthermore, MinCost performs well when the target distance is to be matched as closely as possible. Moreover, Iterative Local Search proves to be a useful post-processing step, since it yields consistent improvements within short computation times.

# Chapter 4
# Conditioned Random Walks

When recording human or animal movement, there are many situations in which the data obtained are sparse, meaning that the recorded location points can be spaced far apart with a significant temporal gap between two measurements. This can be caused by privacy concerns or practical problems such as the battery life[1] or signal reception of GPS trackers. Yet, the unobserved movement between measurements is often of primary interest. For instance, one may want to estimate whether a tracked entity has visited a certain area, a use case that is particularly interesting to validate exposure to diseases in certain regions. Another reason to look at the unobserved movement is to use this to estimate the so-called *home range* of an animal, which is the area in which that animal lives [19] and that it does not normally leave.

This raises the question of how to compute, or estimate, the movement between several discrete points in space-time. The first approach that may come to mind is to connect each point with its successor by a straight line. Such a linear movement model is probably the most commonly used model [65]. Although it provides a reasonable approximation of the movement when locations are measured at a sufficiently high frequency, this is usually not how humans or animals move [89, pp. 36–37] and may lead to incorrect analysis results [18].

One possible solution to this problem is the use of *random walks*. The term was first coined in 1905 by Pearson [68] and refers to a walk that consists of multiple steps, with the direction and length of each step being chosen randomly. In more advanced models, the probabilities for moves in certain directions can vary depending on the direction or other parameters. Random walks allow for the simulation of the movement behavior of an observed entity, providing simple models, which can easily incorporate knowledge about the moving entity (e.g. navigational capacity, behavior) and the environment (e.g., wind, land cover). Random walks require relatively little data to be learned, and are transferable across similar datasets. A large variety of random walk models are used to simulate movement [25], from simple Brownian motion to biased and correlated random walks and more complex models. In particular, state-space models [67] play a prominent role in modeling movement, as they provide means to incorporate external factors and behavioral states into the model.

While these models are frequently used to simulate movement starting at a given location, they cannot readily be used to interpolate between two measurements, since a random walk starting at the location (and time) of the first measurement will most likely be at the wrong location at the time of the second measurement. Existing approaches to generate random walks for interpolation either use simple grid walks or use heuristics

[1] Without affecting their natural behavior, animals can carry at most 5 % of their body weight[83]. As a result, often very small batteries have to be used in GPS trackers.

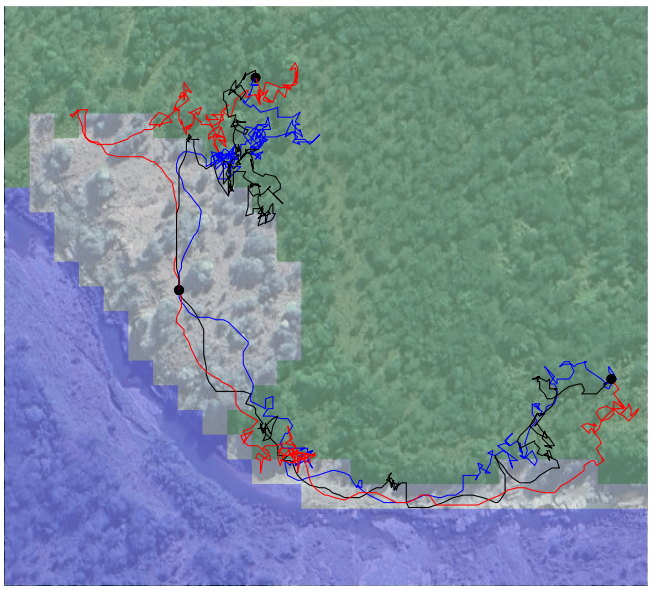


Figure 4.1: Three random walks interpolating between three location measurements (black dots). In the example, the model alternates between Brownian motion (in forest; green) and a correlated random walk (outside forest; brown), and is restricted to the area north of the river (not blue). Dataset: GPS data of a vervet monkey with 1h between measurements [18].

to pull the walk towards the location of the second measurement.

In this chapter, we introduce an efficient way for accurately computing complex random walks for interpolation between two time-stamped location measurements and thereby significantly improve upon existing interpolation methods.

Previous work on movement interpolation using random walk models can be categorized as follows (and are then discussed in more detail in the following):

- For simple random walk models it is possible to compute the spatial distribution of the walks between two points at a given point in time. The most prominent example of this is the *Brownian bridge movement model (BBMM)* [46]. However, being able to compute the distribution is not sufficient to sample random walks. It is also not sufficient to, for instance, compute the probability that a certain region was visited.
- There are several results on even simpler walk models on integer grids [30], in particular for interpolation [56]. The main drawback is the simplicity of the models: in a time unit the walk is limited to stay in the same cell or move to one of its four neighboring grid cells with equal probability.
- For interpolation with more complex random walk models the following approach has been used: interpolated locations are computed by a combination of a random walk step and an attraction force towards the second location [84]. Drawbacks are that it is challenging to set the attraction force appropriately, and that even then the resulting walk does not exactly correspond to the original walk model.
- Many further complex random walk models are frequently used for movement simulation, i.e., movement starting at a given location but with no control over the final location. However, up to now it was not possible to use these for movement interpolation. Such models are further discussed in Section 4.1.4.

Horne, Garton, Krone, and Lewis [46] popularized the *Brownian bridge movement model (BBMM)* as a method of interpolating the movement path of an animal. The BBMM uses a continuous approach utilizing Brownian (i.e., purely random [25]) motion, which is *conditioned* on the given data. The BBMM has been extended in several ways. For example, Benhamou [10] introduced *biased random bridges (BRB)* based on the concept of biased random walks, while Kranstauber, Safi, and Bartumeus [55] introduced bivariate Gaussian bridges, separating the Brownian motion variance into directional components. Brownian bridges and their variants are used for home range estimation, and more generally can be used to estimate spatial (or spatial-temporal) distributions. To a limited degree the BBMM can also be used to compute derived movement parameters such as the distribution of relative speed [18], but these types of bridges do not provide a means to sample individual walks nor to compute visit probabilities or similar.

For home-range estimation, Benhamou [10] also proposed a simplified version of *biased random bridges* (BRB). This approach estimates the utilization distribution from consecutive relocations by linearly interpolating between measurements and, for each interpolated time step, adding a diffusion kernel. In this way, it can describe directed movement over short time intervals while remaining computationally tractable, and thus provides a practical compromise between biological realism and computational efficiency. Benhamou further notes that these diffusion kernels can be adapted to the underlying habitat at the interpolated locations. However, in the simplified BRB approach, movement between two measurements is still approximated via linearly interpolated intermediate points, and habitat information is only used to adapt the kernels placed at these points. As a result, habitat influences the resulting utilization distribution only indirectly, rather than being incorporated directly into the movement model itself. In contrast, our method incorporates habitat directly into the transition model, enabling habitat-dependent interpolation and home-range estimation.

To overcome these limitations, Krumm [56] proposed the concept of *maximum entropy bridgelets*, which are sets of all distinct walks between two vertices in a given time. After computing bridgelets between pairs of location points, Krumm combines them into *bridges*, representing an entire trajectory made up of the (potentially) sparsely sampled points connected by bridgelets. The walks used for maximum entropy bridgelets take steps on a two-dimensional integer grid. In each step, the walk moves to one of the four neighboring grid cells or remains at the current grid cell. However, this method is not efficient, as the computation of one bridgelet requires generating all possible walks between the start and end point, which results in a runtime of $\mathcal{O}(5^T)$ for $T$ time steps. Buchin and Popov [17] show how to make this computation more efficient. Their approach accomplishes computing both the number of walks in a bridgelet and the number of walks visiting a specific point in $\mathcal{O}(T^3)$ time for $T$ time steps [17]. Nonetheless, the random walk model taking steps on the integer grid is rather far from the complex random walk models otherwise used to simulate movement. Similar simple random walk models have been used in *probabilistic time-geography* [97, 58], for instance to estimate visit probabilities [30].

A quite different approach for computing random walks was introduced by Technitis Othman, Safi, and Weibel [83] and Technitis, Weibel, Kranstauber, and Safi [84]. They developed a *random trajectory generation (RTG)* algorithm, which can generate walks based on more complex models. The underlying idea is as follows: At any point in time two components have an effect on the movement. Firstly, according to a random walk model the walk starts at the first location (but is not conditioned under reaching the second location at the right time). Secondly, there is a gravitational/attraction force pulling the walk towards the second location. The combination of these two components guarantees that the second location is reached, while the walk locally has similar characteristics to

the chosen model. However, as also our experiments in Section 4.1.6 confirm, the result of such an approach does not give the same result as generating a walk between the two measured locations according to the given random walk model. Furthermore, setting the gravitational force appropriately is a delicate matter. A similar attraction force is used in [86].

Related to the problem of trajectory interpolation is the problem of trajectory recovery. Trajectory recovery focuses on learning a model for recovering missing locations of trajectories using large datasets [23, 77, 98, 96]. It is not straightforward to see how these methods can be applied to sparser datasets or datasets located in more rural areas. Nor is it clear how these learned models can be used to compute the visit probabilities of particular areas.

**Chapter Structure.** This chapter has two parts. In Section 4.1, we introduce our framework for discretized random walks, enabling efficient computation of transition densities, utilization distributions, and visit probabilities for a broad class of random-walk and state-space movement models. In Section 4.1.6, we evaluate the framework empirically and compare it to existing interpolation approaches. In Section 4.2, we present an applied ecological case study on the Eastern Bettong, demonstrating how the method supports state-dependent habitat-use analysis from sparse GPS trajectories.

## 4.1 Discretized Random Walks

We consider random-walk models specified by one-step transition probabilities between discrete time steps. For uncorrelated models, the state at time $t$ is the grid cell $(x, y)$ and the model is given by a transition matrix $M(i, j) = P\big(X_{t+1} = (x + i, y + j) \mid X_t = (x, y)\big)$ over displacements $(i, j) \in [-R, R]^2$. For correlated and mixed models we augment the state with a discrete heading bin $d \in \{1, \cdots, n_D\}$ and allow the transition matrix to depend on location, time, and previous heading.

There are a variety of random walk models that differ from a “true” random walk, which has the same probability of going in any direction. These different models all introduce some parametrization, allowing us to modify the truly random behavior of a random walk. Codling, Plank, and Benhamou [25] have condensed an overview of some of the most important such models. In the following, we introduce the random walk models used in this chapter.

The most fundamental model, which is sometimes referred to as a simple random walk, generates Brownian motion. In Brownian motion, a walker’s position is described by a continuous-time stochastic process. Furthermore, Brownian bridges are defined by Brownian motion conditioned to start and end points [10].

Another common walk model is the *correlated random walk (CRW)* which is a random walk with the additional property that, for each step, with a fixed probability the last direction will be chosen again [25]. The walk therefore has a correlation between the directions of successive steps. This behavior, called persistence [66], produces a local bias. Correlated random walks can be described using two separate random distributions for the next step’s angle and length [51].

Next, applying a global directional bias to a random walk results in a *biased random walk (BRW)*, which is a random walk with the distinction that it focuses on a certain direction. Thus, in each step that direction has a higher probability of being chosen than the other

directions. This behavior is referred to as a bias. Generally, both the biased direction $d_b$, and the probability $p_b$ with which the bias is applied are fixed over all steps, but there are situations in which a variable bias may be preferable [25].

Movement is influenced by a variety of internal and external factors [64]. State-space models [67] provide a means to incorporate such factors, as the new location of a walker does not only depend on the previous location, but a *state*, which can incorporate for instance direction of movement (e.g. a correlated walk can be seen as a state-space model), the behavioral state of the moving animal or person but also the environment, e.g., land cover or wind speed. Given their generality and ability to incorporate various internal and external factors, it is desirable to use state-space models for interpolation. In the following we present our algorithmic framework to achieve this.

### 4.1.1 Discretizing the Space

Originally, the term random walk refers to a stochastic process describing continuous movement with randomly chosen steps in space and time [33]. In contrast, this chapter focuses on *discrete* random walks, which are obtained by discretizing both space and time. This discretization is essential for enabling dynamic programming–based algorithms and for incorporating spatial constraints such as obstacles or land-cover restrictions. In this section, we formalize how continuous random walk models can be transformed into discrete models that are suitable for efficient computation. In particular, we show how to compute the probability that a random walk starting at a given location ends at a specific location after a fixed number of steps, and the probability that such a walk visits a predefined region during its trajectory.

**Spatial and temporal discretization.** We discretize space by overlaying a regular two-dimensional grid on the plane. Each grid cell is identified by integer coordinates $(x, y) \in \mathbb{Z}^2$. Time is discretized into uniform steps of fixed duration $\Delta t$, indexed by $t \in 0, 1, \cdots, T$.

**Definition 25.** *A* discrete random walk $X$ *is a sequence of grid positions* $[X_0, X_1, \cdots, X_T] = [(x_0, y_0), (x_1, y_1), \cdots, (x_T, y_T)]$ *where* $(x_{t+1}, y_{t+1}) - (x_t, y_t)$ *is a random displacement chosen according to a given random walk model.*

**Discrete transition matrices.** Random walk models are typically defined by continuous probability distributions over step lengths and directions. To obtain a discrete model, we approximate these distributions by a finite set of possible displacements, resulting in a *transition matrix*. A transition matrix $M$ is a matrix of size $(2R+1) \times (2R+1)$ for a maximum step size $R$. The transition matrix is indexed using coordinates $(x, y) \in [-R, R]^2$ with $(0, 0)$ being the center field. Each cell of the matrix contains the probability of making a step from the center to the respective cell (see Figure 4.2 for an example transition matrix for Brownian motion). Furthermore, by construction, the transition matrix satisfies

$$\sum_{(x,y)\in[-R,R]^2} M[x, y] = 1.$$

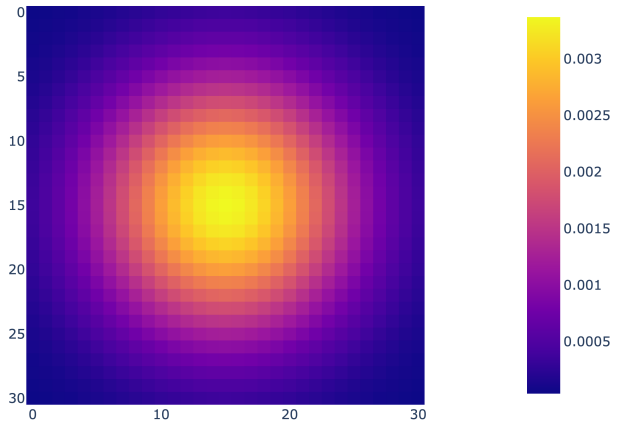



Figure 4.2: A heatmap showing a transition matrix for modeling Brownian motion with $R = 15$.

### Transition Density

Using the transition matrix $M$, we compute the *transition density* $D$, i.e., the probability distribution over grid cells after a given number of time steps. Concretely, for a fixed start cell $(x_0, y_0)$, $D(x, y, t)$ denotes the probability that the random walk is located at cell $(x, y)$ after exactly $t$ time steps.

We compute density $D$ by dynamic programming, extending the approach of Buchin and Popov [17]. Let the spatial grid be indexed by $(x, y) \in \{0, \cdots, W - 1\} \times \{0, \cdots, H - 1\}$ and time by $t \in \{0, \cdots, T\}$. The transition matrix $M$ has size $(2R + 1) \times (2R + 1)$ and is indexed by displacements $(i, j) \in [-R, R]^2$, where $M(i, j)$ is the probability of moving by displacement $(i, j)$ in one time step.

**Boundary handling.** All computations are carried out on a finite grid window. Any probability mass that would step outside the window is ignored. After each time step we renormalize the probability layer so that it sums to one. Intuitively, this means we simulate the walk only while it remains inside the modeled window.

**Dynamic program.** The following recursive function can be used to calculate the transition density:

$$D(x, y, t) = \begin{cases} 1 & \text{if } (x, y) = (x_0, y_0) \wedge t = 0, \\ 0 & \text{if } (x, y) \neq (x_0, y_0) \wedge t = 0, \\ \sum_{(i,j) \in [-R,R]^2} D(x-i, y-j, t-1) M(i, j) & \text{otherwise.} \end{cases} \tag{4.1}$$

To illustrate this, an example of a simple transition matrix application can be seen in Figure 4.3. In the figure, the probability of being at the field $(x, y)$ at time step $t$ is computed. Take the northern neighbor $(x, y - 1)$ of $(x, y)$ as an example. For a walk that reached that neighbor after $t - 1$ steps to reach $(x, y)$ one step later, a step to the south must be made. Therefore the value 0.2 from the matrix's *southern* field is taken. Thus, the probability of reaching $(x, y - 1)$ at $t - 1$ and then transitioning to $(x, y)$ at $t - 1 \to t$, can be computed as $D(x, y - 1, t - 1) \cdot M(0, 1) = 0.5 \cdot 0.2 = 0.1$. The probabilities of visiting $(x, y)$ after $t$ steps

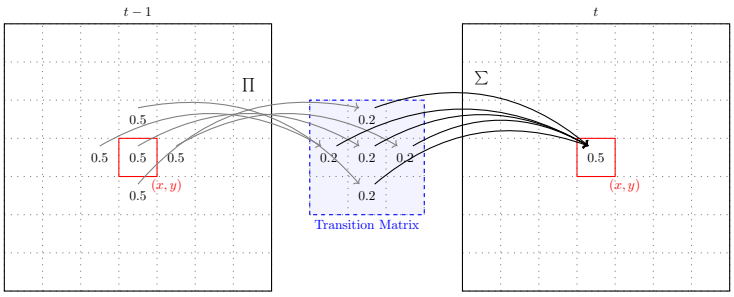



Figure 4.3: The probability of a field $D(x, y, t)$ is computed by applying a transition matrix with $R = 1$ using multiplication and then adding up the probabilities for all directions.

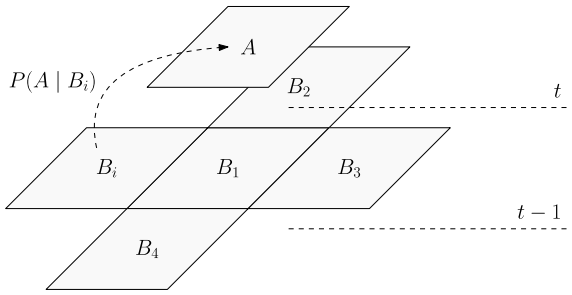



Figure 4.4: The consecutive time steps computed by the dynamic program can be visualized as layers stacked on top of each other. The probability of transitioning from a field $B_i$ at $t - 1$ to a field $A$ at $t$ is $P(A \mid B_i)$.

coming from all other neighbors are computed similarly. Finally, the sum of all of those probabilities is calculated and taken as the value of $D(x, y, t)$.

**Forward and backward transition probabilities.** The computation of each field's probability can be derived more formally as follows. Figure 4.4 shows some fields of two arbitrary time steps $t - 1$ and $t$. Assume that the dynamic program is computing the probabilities at time step $t$, more specifically the probability $P(A)$ of being at the field $A$ at time step $t$. As can be seen in the figure, the probability for transitioning from a field $B_i$ at $t - 1$ to field $A$ at $t$ is the conditional probability $P(A \mid B_i)$. In practice, these transitioning probabilities are provided by the transition matrix. As described above, the probability $P(A)$ of being at $A$ at time step $t$ is calculated as the probability of being at any of its neighboring fields at time step $t - 1$ and then transitioning to $A$ at $t - 1 \to t$. Thus, $P(A)$ is

$$P(A) = \sum_{B_i \in \mathcal{N}_A} P(B_i) \cdot P(A \mid B_i) \tag{4.2}$$

with $\mathcal{N}_A$ being the neighbors of A. Given these probabilities, another important probability can be derived, which becomes relevant when generating actual random walks later on: The probability $P(B_i \mid A)$ of choosing a field $B_i$ as the next step for the walk when coming from field $A$. As $P(A \mid B_i)$ is already known, $P(B_i \mid A)$ can easily be calculated using Bayes' theorem [39] as

$$P(B_i \mid A) = \frac{P(A \mid B_i) \cdot P(B_i)}{P(A)}. \tag{4.3}$$

**Visit Probability**

In many applications we are not only interested in the probability of *ending* at a given cell, but also in the probability that a walk *visited* a target region at least once. Recall that $X = \{X_0, \cdots, X_T\}$ denotes the (discrete) random walk on the grid induced by the transition matrix $M$ and start cell $X_0 = (x_0, y_0)$ as in Section 4.1.1. Let $C_\tau$ be a (possibly time-dependent) target region (set of cells) at time $\tau$. We define the visit event

$$\text{VIS}_t := \left\{\exists \tau \in \{0, \cdots, t\} : \ X_\tau \in C_\tau\right\}.$$

For any cell $(x, y)$ and time $t$, we want to compute the conditional probability

$$D_v(x, y, t) \ := \ P\big(\text{VIS}_t \,\big|\, X_t = (x, y)\big),$$

i.e., the fraction of walks that end in $(x, y)$ at time $t$ and have visited the target region at least once.

**Recurrence.** We augment the transition density matrix $D$ such that each cell stores a second value which will contain the fraction of walks that have visited at least one of the cells in target area $C_t$. For ease of notation, this augmented value can be accessed by calling $D_v$, in other words $D_v(x, y, t)$ contains the fraction of paths starting in cell $(x_0, y_0)$ that have visited at least one of the cells in target area $C_\tau$ and end in cell $(x, y)$ after $t$ time steps. Furthermore, we define $D_v(x, y, t) = 0$ whenever $D(x, y, t) = 0$.

$$D_v(x, y, t) = \begin{cases} 1 & \text{if } (x, y) \in C_t \\ 0 & \text{if } (x, y) \notin C_t \wedge t = 0 \\ \frac{\sum_{(i,j)\in[-R,R]^2} D_v(x-i, y-j, t-1) \cdot D(x-i, y-j, t-1) \cdot M(i,j)}{D(x,y,t)} & \text{otherwise (and } D(x, y, t) > 0) \end{cases} \tag{4.4}$$

If $(x, y) \in C_t$, then every path ending at $(x, y)$ at time $t$ has (at the latest at time $t$) visited the target, hence $D_v(x, y, t) = 1$. For $t = 0$, only the start cell is reachable; therefore $D_v(x_0, y_0, 0) = 1$ if the start cell lies in $C_0$ and 0 otherwise.

For the remaining case, consider all predecessor cells $(x - i, y - j)$ at time $t - 1$ from which the walk can move to $(x, y)$ in one step with probability $M(i, j)$. The probability mass of paths that end at $(x - i, y - j)$ at time $t - 1$ and have visited the target by then equals $D_v(x - i, y - j, t - 1) \cdot D(x - i, y - j, t - 1)$. Multiplying by $M(i, j)$ gives the probability mass of such paths that additionally take the step to $(x, y)$. Summing over all displacements yields the total probability mass of paths that end in $(x, y)$ at time $t$ and have visited the target by time $t$ (note that in this case $(x, y) \notin C_t$, so visiting at time $t$ cannot be the first visit). Finally, dividing by $D(x, y, t)$, the probability mass of all paths ending in $(x, y)$ at time $t$, gives us the desired fraction, i.e. the conditional probability $P(\text{Vis}_t \mid X_t = (x, y))$.

## 4.1.2 Movement Interpolation and Utilization Distribution

Using the transition density as described in Section 4.1.1, we can now efficiently interpolate random walks, conditioned to a start and end point, utilizing different random walk models. Beyond sampling individual conditioned walks, we can compute a start–end conditioned utilization distribution, i.e., the expected occupancy probability over space and time induced by all feasible paths.

**Algorithm 1:** The WALKER algorithm used to generate walks.

**Data:** transition density $D$, end point $(x, y)$, time limit $T$, step length $R$, transition matrix $M$
**Result:** walk $W$

$W \leftarrow$ empty array
**for** $t \leftarrow T$ **downto** $1$ **do**
  $W \leftarrow (x, y) \cup W$
  $N \leftarrow$ neighbors of $(x, y)$ in range $[-R, R]^2$
  $P \leftarrow \left[\frac{D(x-i,y-j,t-1)M(i,j)}{D(x,y,t)} \mid (i, j) \in N\right]$
  $l \leftarrow$ choose random location weighted by the probabilities in $P$
  $(x, y) \leftarrow$ field in chosen location $l$
$W \leftarrow (x, y) \cup W$

#### Efficient Movement Interpolation

To accomplish the movement interpolation we describe an algorithm that reconstructs walks using the data computed by the dynamic program. Such an algorithm, which we call a *walker* in the following, was described by Buchin and Popov [17]. We extend their algorithm to support our modified dynamic programming approach and larger step sizes. If there is a path possible from the start to end location, the WALKER algorithm will always generate a possible random walk.

The WALKER algorithm reconstructs a walk by generating the steps of the walk starting at the end point and backtracking towards the start point. This backwards construction has the major advantage that a walk can always be generated that traverses from start point to end point (as long as such a walk exists). For each step that is generated, the algorithm computes the probability of going from the current to any neighboring field using Equation 4.3. Algorithm 1 shows the algorithm in pseudocode.

The algorithm takes four arguments, specifically the transition density $D$ previously computed by a dynamic program, the end point for the walk at $(x, y)$, the number of steps $T$ that the walk should have, and the transition matrix $M$ used for computation of the dynamic program. For each step, the current position is added to the walk. Afterwards, all possible neighbors of $(x, y)$ are collected. For each neighboring field, the probability of transitioning to that field is computed. These conditional probabilities are computed using 4.3 as explained above. As the walk is constructed backwards, the next step is queried by using $t - 1$, instead of $t + 1$. The next field's direction is chosen randomly by taking the probability of each field into account as a custom random distribution. Thereafter, the chosen field is set as the next field of the walk.

#### Utilization Distribution

In many ecological applications, one is not only interested in sampling plausible paths between two measurements, but also in the *spatio-temporal intensity* induced by *all* paths that are consistent with the measurements. This is commonly captured by a *utilization distribution (UD)*. Informally, the UD assigns to each cell the probability mass (or expected occupancy) that the interpolated movement spends in that cell.

This quantity differs from the visit probability in Section 4.1.1: $\mathrm{UD}(x, y, t)$ describes *where the walk is at time t* under the conditioning, whereas visit probabilities describe whether

a region was hit *at least once* at any time up to $t$. Aggregating UD over time yields an *expected-occupancy* map, while visit probabilities yield *hitting* probabilities.

We compute the UD conditioned to a start and end point efficiently via a forward and backward decomposition. First, we calculate the forward transition density $D_f$ by dynamic program as described in Equation 4.1. Then, we additionally calculate the backward transition density $D_b$, where $D_b(x, y, t)$ denotes the probability that a random walk starts at cell $(x, y)$ and ends in $(x_T, y_T)$ after exactly $T - t$ time steps. Intuitively, $D_b$ propagates probability mass backward from the endpoint. When we step backward by $(i, j)$, this corresponds to a forward step by $(-i, -j)$, so the recurrence uses the reversed displacement probabilities $M(-i, -j)$. Then, we can calculate $D_b$ by a dynamic program that mirrors Equation 4.1:

$$D_b(x, y, t) = \begin{cases} 1 & \text{if } (x, y) = (x_T, y_T) \wedge t = T, \\ 0 & \text{if } (x, y) \neq (x_T, y_T) \wedge t = T, \\ \sum_{(i,j)\in[-R,R]^2} D_b(x + i, y + j, t + 1)M(i, j) & \text{otherwise,} \end{cases} \tag{4.5}$$

with the same boundary convention as before (zero padding outside the grid).

**Computing the UD.** We want the cell $\mathrm{D_U}(x, y, t)$ to be the conditioned utilization distribution, i.e. what percentage of paths from $(x_0, y_0)$ to $(x_T, y_T)$ do go through cell $(x, y)$ at exactly time $t$. Hence the conditioned utilization distribution is

$$\mathrm{D_U}(x, y, t) = \frac{D_f(x, y, t)\, D_b(x, y, t)}{D_f(x_T, y_T, T)}. \tag{4.6}$$

For each fixed $t$, $\sum_{x,y} \mathrm{UD}(x, y, t) = 1$ (provided $D_f(x_T, y_T, T) > 0$).

**Time-aggregated utilization distribution.** Often one wants a single spatial map rather than time-separated slices of the utilization distribution. A standard aggregation is the expected fraction of time spent in each cell:

$$\mathrm{UD}(x, y) := \frac{1}{T + 1} \sum_{t=0}^{T} \mathrm{UD}(x, y, t). \tag{4.7}$$

Equivalently, $(T + 1) \cdot \mathrm{UD}(x, y)$ equals the expected visit duration of cell $(x, y)$ by the conditioned walk. This representation is directly comparable to home-range style maps, while still respecting the start/end conditioning.

### 4.1.3 Correlated and Mixed Random Walks

The constructions and algorithms described in Section 4.1.1 and 4.1.2 work for simple random walk models like Brownian and biased random walks. However, the aforementioned constructions can be extended to model correlated random walks, or walks that change the model based on temporal or spatial data.

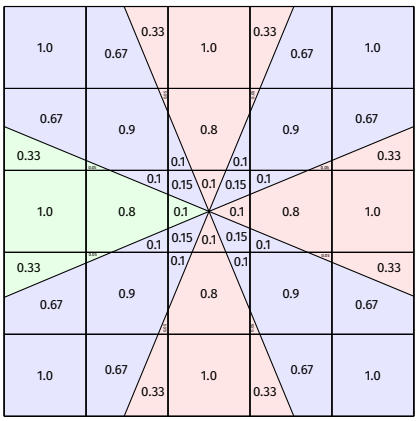



Figure 4.5: A matrix where for every $1 \le d \le 8$ each cell that overlaps with a wedge $\Theta_d$ is contained into set of cells $C_d$. For example, all cells that overlap with the green slice are in set $C_1$. Moreover, overlap weights are shown for each occurrence of a cell in the corresponding set $C_d$.

### Correlated Random Walks

In correlated random walks there is persistence in the direction of movement. In literature this is often modeled by choosing a turning angle out of a normal distribution with the mean turning angle continuing in the same direction as the previous step. However, this process is not possible with our discretized model as the transition density does not store where paths are coming from. Moreover, as with the other variables, we need to discretize the turning angles.

**Discretizing turning angles.** First, we discretize the turning angle into $n_D$ distinct discrete directions. Therefore, we partition the angle range $[0, 2\pi)$ into $n_D$ direction bins

$$\Theta_d = \left[\frac{2\pi(d-1)}{n_D}, \frac{2\pi d}{n_D}\right], \qquad d \in \{1, \cdots, n_D\}.$$

Intuitively, each bin corresponds to a wedge (sector) emanating from the origin. Any grid displacement $(\Delta x, \Delta y)$ induces a continuous heading $\theta(\Delta x, \Delta y) = \text{atan2}(\Delta y, \Delta x)$, which we map to the unique bin $d$ with $\theta(\Delta x, \Delta y) \in \Theta_d$.

Because the grid is coarse, a single displacement cell may overlap multiple wedges. Therefore, rather than using a hard assignment, we define *overlap weights*

$$w_d(\Delta x, \Delta y) \in [0, 1],$$

equal to the fraction of the cell area covered by sector $\Theta_d$ (and normalized such that $\sum_{d=1}^{n_D} w_d(\Delta x, \Delta y) = 1$ for each displacement cell). Figure 4.5 visualizes these weights: the numbers inside cells indicate how strongly a cell belongs to a wedge (full or partial overlap). Using these weights, the set of displacements consistent with direction $d$ is

$$C_d = \{(\Delta x, \Delta y) \in [-R, R]^2 \ : \ w_d(\Delta x, \Delta y) > 0\},$$

with $w_d$ providing a "soft membership" near wedge boundaries.

**Correlated transition density.** In order to keep track of which direction a walk came from in the last step we add an extra dimension to the transition density $D'$ which is a spatial grid indexed by $(x, y) \in \{0, \cdots, W-1\} \times \{0, \cdots, H-1\}$, direction $d \in \{1, \cdots, n_D\}$ and time by $t \in \{0, \cdots, T\}$. The value in a cell $D'(x, y, d, t)$ contains the probability a path ends in cell $(x, y)$ at time $t$ where the last step has a heading that belongs to direction bin $\Theta_d$.

### Mixed Random Walks

Furthermore, the discretized construction allows for dynamic transition matrix selection based on spatial or temporal data. Let $d$ denote the direction index. Hence, we define $M_{x,y,d,t}$ to be the transition matrix which should be applied at location $x, y$, at time step $t$, for direction $d$. In the purely spatial case, this reduces to $M_{x,y,d}$, and in the purely temporal case to $M_{d,t}$. If $d$ exceeds the number of directions defined for the corresponding location or time step, we apply the empty transition matrix. This may happen, for example, when combining a Brownian motion transition matrix with a correlated-motion transition matrix.

### Forward transition density for mixed and correlated walks

The forward recurrence generalizes Equation 4.1 by carrying direction $d$ and using the locally selected kernel $M_{x,y,d,t}$.

We can now update Equation 4.1 to work with mixed and correlated random walks.

$$D'(x, y, d, t) = \begin{cases} 1/n_D & \text{if } (x, y) = (x_0, y_0) \wedge t{=}0, \\ 0 & \text{if } (x, y) \neq (x_0, y_0) \wedge t{=}0, \\ \sum_{d'=1}^{n_D} \sum_{(i,j) \in C_d} w_d(i, j) M_{x-i,y-j,d',t-1}(i, j) D'(x-i, y-j, d', t-1) & \text{otherwise.} \end{cases} \tag{4.8}$$

Alternatively, when we know the initial direction $d_0$ of the first measurement we set $D'(x_0, y_0, d_0, 0) := 1$ and $D'(x, y, d, 0) := 0$ for all $(x, y, d) \neq (x_0, y_0, d_0)$.

### Visit probabilities for mixed and correlated walks

Analogously, we generalize Equation 4.4 to also work with the mixed and correlated walks.

$$D'_v(x, y, d, t) = \begin{cases} 1 & \text{if } (x, y) \in C_t, \\ 0 & \text{if } (x, y) \notin C_t \wedge t = 0, \\ \dfrac{\sum_{d'=1}^{n_D} \sum_{(i,j) \in C_d} D'_v(x-i,y-j,d',t-1) \cdot w_d(i,j) M_{x-i,y-j,d',t-1}(i,j) D'(x-i,y-j,d',t-1)}{D'(x,y,d,t)} & \text{otherwise} \\ & \text{and } D'(x, y, d, t) > 0. \end{cases} \tag{4.9}$$

### Utilization distribution for mixed and correlated walks

We define the backward density $D'_b(x, y, d, t)$ as the probability that a walk which is at cell $(x, y)$ at time $t$ with heading bin $d$ reaches the target cell $(x_T, y_T)$ at time $T$ under the same

mixed/correlated dynamics. Intuitively, $D'_b$ propagates probability mass backward from the endpoint (analogous to Equation (4.5)).

To write the recurrence, recall that in the forward update (Equation (4.8)) a transition into heading bin $d$ is formed by summing over predecessor heading bins $d'$ and predecessor displacements $(i,j) \in C_d$, weighted by the overlap weights $w_d(i,j)$ and the locally selected kernel evaluated at the predecessor state. Mirroring this construction yields the following backward dynamic program with the same boundary convention as before (zero padding outside the grid):

$$D'_b(x,y,d,t) = \begin{cases} 1 & \text{if } (x,y) = (x_T, y_T) \wedge t = T, \\ 0 & \text{if } (x,y) \neq (x_T, y_T) \wedge t = T, \\ \sum_{d'=1}^{n_D} \sum_{(i,j)\in C'_d} w_{d'}(i,j) M_{x,y,d,t}(i,j) D'_b(x{+}i, y{+}j, d', t{+}1) & \text{otherwise.} \end{cases} \tag{4.10}$$

And similarly:

$$\mathrm{D}'_{\mathrm{U}}(x,y,d,t) = \frac{D'_f(x,y,d,t)\, D'_b(x,y,d,t)}{\sum_{d'=1}^{n_D} D'_f(x_T, y_T, d', T)}. \tag{4.11}$$

For each fixed $t$, $\sum_{x,y} \mathrm{UD}(x,y,t) = 1$ (provided $D_f(x_T, y_T, T) > 0$).

Often one wants a single spatial map rather than time-separated slices of the utilization distribution. A standard aggregation is the expected fraction of time spent in each cell:

$$\mathrm{UD}(x,y) := \frac{1}{T+1} \sum_{t=0}^{T} \sum_{d=1}^{n_D} \mathrm{D}'_{\mathrm{U}}(x,y,d,t). \tag{4.12}$$

### 4.1.4 Movement models

**Brownian Walks.** First, we introduce a transition matrix that is designed to model Brownian motion, therefore discretely approximating Brownian bridges. To achieve this, the transition matrix $M_{\text{brownian}}$ can be computed from a normal distribution by computing the probability for each field $x = (i,j)$ using the *probability density function (PDF)* $f_{\text{mnorm}}(\mu, \Sigma, x)$ of the multivariate normal distribution [47, p. 150]:

$$\begin{aligned} M'_{\text{brownian}}(\mu, \Sigma) &= \left[ f_{\text{mnorm}}(\mu, \Sigma, x = (i,j))_{ij} \right]_{(i,j)\in[-R,R]^2} \\ M_{\text{brownian}}(\mu, \Sigma) &= \text{normalize}(M'_{\text{brownian}}(\mu, \Sigma)). \end{aligned} \tag{4.13}$$

with $\mu$ being the expected value, and $\Sigma$ being the covariance matrix of the distribution. We first compute the matrix $M'_{\text{brownian}}$ and normalize it afterwards to ensure that the sum of all cells of the matrix equals 1 (see Figure 4.2). Normalization is achieved by applying the function

$$\text{normalize}(M) = \frac{1}{\sum_{(i,j)\in[-R,R]^2} M_{i,j}} \cdot M. \tag{4.14}$$

**Correlated Random Walks.** Next, we introduce a transition matrix for correlated random walks. This matrix is based on the idea of Kareiva and Shigesada [51] to describe

correlated random walks using two separate distributions for the angle and length of each step in a walk. For choosing the distributions, we follow the implementation of unconditioned correlated random walks by Calenge in their R library *adehabitatLT* [20]. We define two separate transition matrices for both distributions and subsequently combine them by multiplying their values. First, we define the matrix modeling the step length distribution as

$$\begin{aligned} M'_{\text{length}} &= \left[ f_{\text{chi}}(k = 2, x = d((0,0),(i,j))) \right]_{(i,j)\in[-R,R]^2} \\ M_{\text{length}} &= \text{normalize}\left(M'_{\text{length}}\right) \end{aligned} \tag{4.15}$$

with $d(p,q)$ being the *Euclidean distance* between two points $p$ and $q$, and $f_{\text{chi}}(k,x)$ being the PDF of the chi distribution. The parameter $k$ describes the degrees of freedom of the distribution. Following this, we define the transition matrix modeling the angle distribution as

$$\begin{aligned} M'_{\text{angle}} &= \left[ f_{\text{wrpnorm}}(\mu = 0, \rho = 0.9, x = \alpha((i,j)))_{ij} \right]_{(i,j)\in[-R,R]^2} \\ M_{\text{angle}} &= \text{normalize}(M'_{\text{angle}}) \end{aligned} \tag{4.16}$$

with $f_{\text{wrpnorm}}$ being the wrapped normal distribution [48], and $\alpha(p)$ being the angle of the point $p$ to the vector $(1,0)$. We choose the values for $\mu$ and $\rho$ based on the implementation of the RTG algorithm by Technitis et al. [83] We define $f_{\text{wrpnorm}}$ as follows based on the implementation in the *CircStats* library [60], which is used by the adehabitatLT library:

$$f_{\text{wrpnorm}}(\mu,\rho,x) = f_{\text{norm}}(\mu, \sigma = \sqrt{-2\log\rho}, x) \mod 2\pi. \tag{4.17}$$

What remains is to combine both matrices to get $M_{\text{crw}}$ as follows:

$$M_{\text{crw}} = \text{normalize}(M_{\text{length}} \cdot M_{\text{angle}}). \tag{4.18}$$

**Biased Random Walks.** Finally, we use a transition matrix for approximating biased random walks as inspired by Benhamou [10]. These walks have a fixed persistence in a given direction $d$, hence we can simply obtain a biased transition matrix $M_{\text{bias}}$ by rotating matrix $M_{\text{crw}}$ to face direction $d$.

### 4.1.5 Sum of Gaussians

Deciding on a model can be hard or near impossible with real-world applications such as animal movement. Hence, one might want to use a sum of Gaussians to determine transition matrices. In real movement data, animals often exhibit several distinct movement modes, such as short, irregular steps during foraging and longer, more directed steps during traveling. Rather than committing to a single parametric walk model, we capture this behavior by constructing transition matrices directly from observed displacements using a sum of Gaussian components.

Let $(x_t, y_t)$ denote observed locations sampled at a constant time interval $\Delta t$. From these locations we derive step displacements

$$(\Delta x_t, \Delta y_t) = (x_{t+1} - x_t,\ y_{t+1} - y_t).$$

We model the empirical distribution of these displacements as a weighted sum of $K$

Gaussian distributions,

$$p(\Delta x, \Delta y) = \sum_{m=1}^{K} \pi_m \mathcal{N}((\Delta x, \Delta y) \mid \mu_m, \Sigma_m),$$

where each component represents a characteristic movement mode with mean displacement $\mu_m$ and variability $\Sigma_m$, and the weights $\pi_m$ reflect how frequently each mode occurs in the data. The parameters of this mixture can be estimated from the observed step displacements.

To obtain a discrete transition matrix, we fix a maximum step size $R$ and discretize the continuous displacement distribution onto the grid. For each displacement $(i, j) \in [-R, R]^2$, we define the unnormalized transition weight

$$M'_{\text{mix}}(i, j) = \sum_{m=1}^{K} \pi_m f_{\text{mnorm}}(\mu_m, \Sigma_m, x = (i, j)),$$

where $f_{\text{mnorm}}$ denotes the density of the bivariate normal distribution. The final transition matrix $M_{\text{mix}}$ is obtained by normalization so that the probabilities over all possible displacements sum to one.

### 4.1.6 Evaluation

In this section we evaluate different random walk models that can be approximated with our discretized method. All experiments in this chapter were conducted on a machine equipped with an Intel Core i9-14900K processor featuring 24 cores and 32 hardware threads with a maximum clock frequency of 6.0 GHz. The system has 64 GB of RAM. The source code for the experiments is available on GitHub[2].

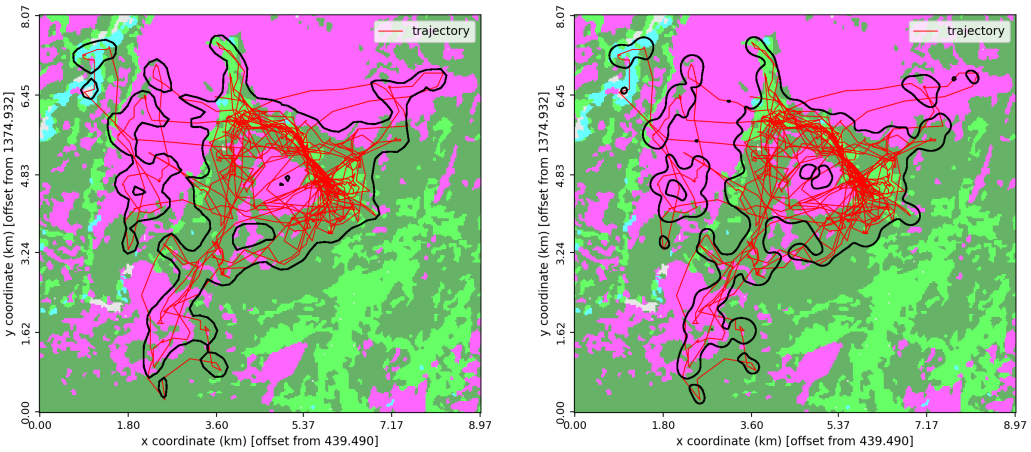
#### Home Range Estimation

We start by comparing the simplified BRB approach of Benhamou [10] with our method. To keep the runtimes manageable we project the trajectory to a coarser grid with cell size 30 m (which is the same as the resolution of data containing habitat parameters). Benhamou uses as a diffusion coefficient of $D = 440\,\text{m}^2/\text{min}$, which we use for both methods. This corresponds to a standard deviation of $\sigma = \sqrt{2D\tau} = 29.7$ m for a time step of $\tau = 1$ min, which we additionally map to the grid by dividing by the cell size of 30 m, yielding $\sigma = 0.99$ in grid units. Moreover, Benhamou uses a minimum relocation standard deviation of $s_{\text{min}} = 100$ m, intended to encompass alternative locations within the same habitat patch, and we adopt this value for both methods as well. Figure 4.6 shows example home-range estimates for the same trajectory and with the same parameter settings. In both plots, the red polyline shows the observed trajectory and the black contour shows the estimated home range overlaid on the habitat map.

Benhamou also discusses diffusion parameters for specific habitats, however they say these have to be considered carefully as these coefficients stem from a small number of datapoints. Nonetheless, for demonstration purposes Figure 4.7 shows the home range estimate as calculated with habitat dependent diffusion parameters.

Qualitatively, the three estimates show a trade-off between smoothness and accurate

[2] https://github.com/traMectory/random-walks-python




(a) Home range estimate with simplified BRB.

(b) Home range estimate with our method.

Figure 4.6: Example home range estimates overlaid on the habitat map: (a) Biased Random Bridges (BRB) and (b) our discretized random-walk method. Both methods were run with the same parameter settings ($\sigma_{\text{min}} = 100$, $D = 440\,\text{m}^2/\text{min}$, $\tau = 60\,\text{s}$). The red lines show the trajectory and the black contour indicates the estimated home range.

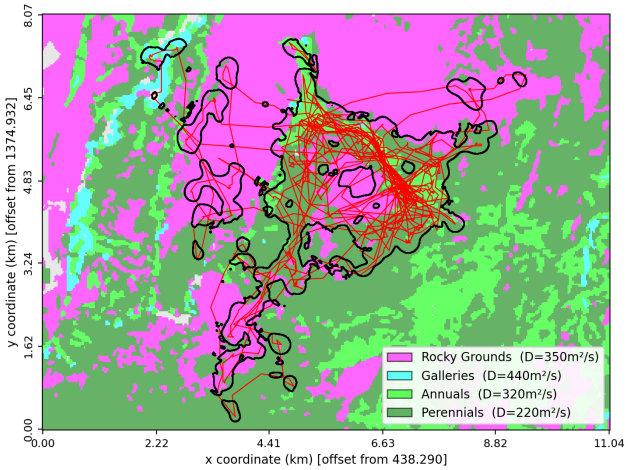
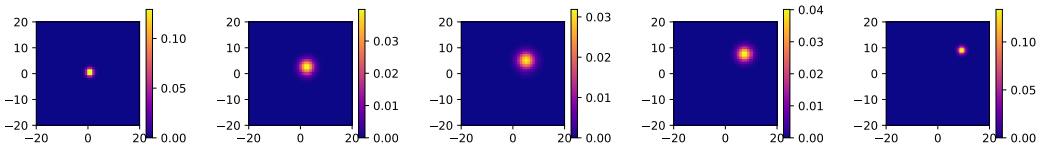
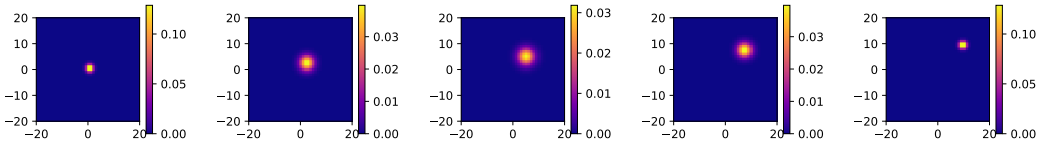



Figure 4.7: Example home range estimate of our methods with the habitat parameters from [10] ($s_{\text{min}} = 100$, $\tau = 60\,\text{s}$). The red lines show the trajectory and the black contour indicates the estimated home range.



(a) $t = 1$ (b) $t = 5$ (c) $t = 10$ (d) $t = 15$ (e) $t = 19$

Figure 4.8: Density plots of Brownian Walks generated with our method at different time slices with a standard deviation of 1, traveling from (0, 0) to (10, 10) in 20 time steps.



(a) $t = 1$ (b) $t = 5$ (c) $t = 10$ (d) $t = 15$ (e) $t = 19$

Figure 4.9: Density plots of Brownian Walks at different time slices with a standard deviation of 1, traveling from (0, 0) to (10, 10) in 20 time steps.

movement modeling. The simplified BRB estimate (Figure 4.6a) is the smoothest. Our estimate with uniform parameters (Figure 4.6b) is less smooth, but still relatively coherent. The habitat-dependent estimate in Figure 4.7 is the least smooth, but it is the most consistent with the specified habitat parameters, because habitat is incorporated directly into the transition model. Since Benhamou [10] notes that these habitat parameters are difficult to estimate reliably from limited data, we do not interpret this comparison as a validation of parameter correctness, but rather as an illustration of how parameter choices affect the resulting home-range estimate.

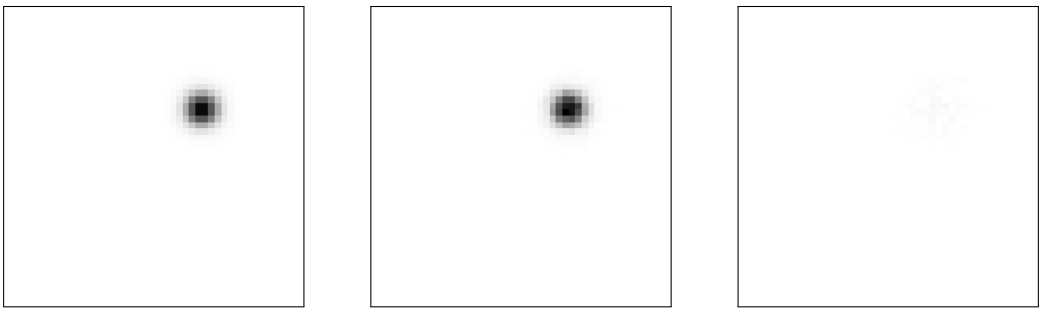
**Runtime.** Another trade-off is computation time: the simplified BRB method is significantly faster. On the habitat raster of size 368 × 409, the simplified BRB method takes on average 0.43 s to compute the home-range estimate for the full trajectory. In contrast, our method takes substantially longer, because it computes segment-wise conditioned random-walk interpolation with habitat-dependent transitions.

These timings exclude plotting and include only the computation of the home-range estimate itself. While the simplified BRB method is evaluated on the full raster, our method processes the trajectory segment-wise on local cropped windows extracted from the habitat map. In total, we process 1303 segments. For a segment with endpoints $(r_0, c_0)$ and $(r_1, c_1)$, we take the bounding box of the two endpoints and enlarge it by a padding of 30 cells in each direction (clipped to the raster boundary). We use `step_size`=15 and `padding`=30 in all experiments. The resulting local windows have mean dimensions of 65.03 × 65.18 cells.

For our method, the mean runtime per segment is 1.05 s, which yields a total runtime of 1371.11 s (approximately 22.9 minutes) for the full trajectory.

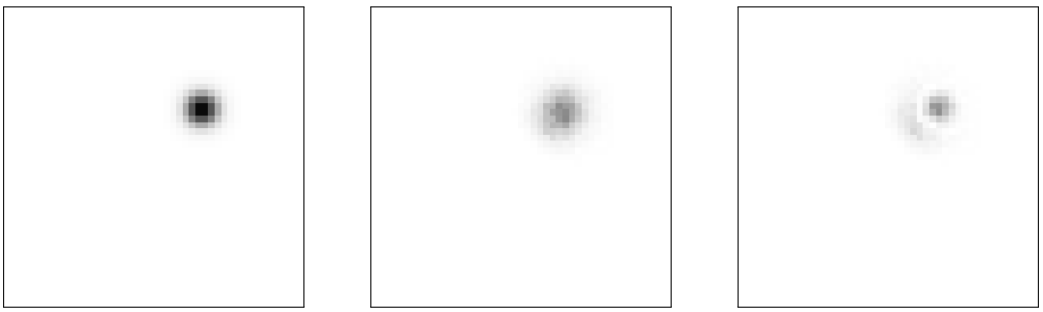
### Brownian Walks

With Brownian bridges, given a start point $s$, end point $q$, time steps $T$, and diffusion coefficient $\sigma^2$, the expected position of an animal moving randomly between $s$ and $q$ can be



(a) Analytical density function (b) Density function as generated by our method (c) Absolute difference

Figure 4.10: The difference between the analytical ground truth of Brownian motion and the density function of $10^6$ random walks as generated with our method.

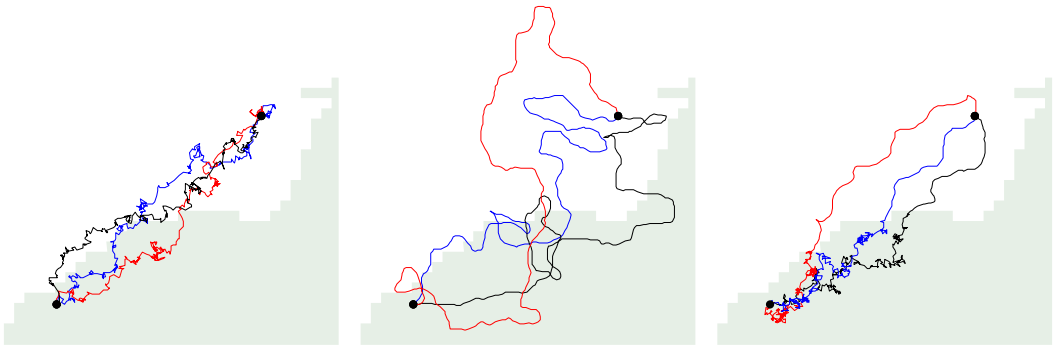


(a) Analytical density function (b) Density function as generated by the RTG method (c) Absolute difference

Figure 4.11: The difference between the analytical ground truth of Brownian motion and the density function of $10^6$ random walks as generated with the RTG algorithm.

estimated analytically [46]. This analytical estimation can be seen as the ground truth of this Brownian motion. Figure 4.9 shows the occupation probability of a Brownian bridge where $s = (0, 0)$, $q = (10, 10)$, $T = 20$, and $\sigma = 1$ for different time slices. Figure 4.8 shows the quantitative occupation probabilities of Brownian walks generated by our method with the aforementioned parameters. These occupation probabilities were calculated by generating $10^6$ walks and counting how often a cell is occupied at time step 15. Furthermore, we can compare the occupation probabilities of our method with the analytical occupation probabilities by comparing the values of each cell, as is shown in Figure 4.10. Figure 4.10c shows that our method closely matches the ground truth for the given parameters. Similarly, in Figure 4.11 Brownian bridges generated by the RTG algorithm are compared with the analytical ground truth. The RTG heuristically pulls walks towards the end point, and, as seen in Figure 4.11c these generated occupation probabilities do not match the analytical occupation probabilities.

Additional examples of Brownian motion generated with our method can be found in Figure 4.12a and in the middle sections of Figures 4.13a and 4.13b.



(a) Brownian motion (b) Correlated motion (c) Field type aware motion

Figure 4.12: Examples of mixed random walks on spatial data, with $T = 300$, $D_{max} = 16$, and a transition matrix size of 15.

### Correlated Random Walks

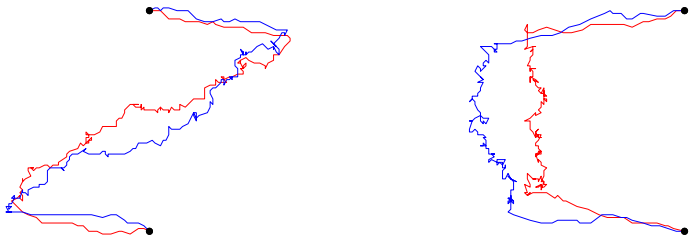
Figure 4.14 shows a comparison of correlated random walks with $T = 300$ time steps generated with the RTG algorithm and our method. For the correlated random walks in Figure 4.14b a transition matrix $M_{\text{crw}}$ with size $|M_{\text{crw}}| = 15$ was with $D_{max} = 16$. An additional example with the same parameters can be found in Figure 4.12b.

### Mixed Walks

The framework also allows bridges in which movement models are switched based on spatial or temporal data. Examples for mixed random walks on spatial data can be seen in Figures 4.1 and 4.12c. Figure 4.1 uses location measurements of a vervet monkey from the vervet monkey database [18]. This vervet monkey dataset consists of only 454 data points in an area of 2.19 km by 1.77 km. All points were recorded over a time span of 31 days with points taken hourly each day from 3:00 am to 5:00 pm, with 11 measurements missing. Figure 4.1 shows an area of approximately 300 x 300 meters in size, and contains in total 45 of the data points (including the three points shown). For the spatial data used in the experiments we use land coverage data from the Esri Sentinel-2 10-Meter Land Use/Land Cover dataset [72]. The dataset partitions areas of the world into fields of 10 by 10 meters in size. These partitions are then classified into nine classes. In our examples, we highlight the forest class in green, the rangeland class in white, and the water class in blue. The land coverage data is provided in the GeoTIFF file format that we had to convert to the format we use in the framework for the experiments. For a dynamic program with time limit $T$, the framework takes an array of size $W \times H$ that contains a land type for each field. We extracted the required area from the dataset and converted the coordinates to the *EPSG:32736* format, which provides a local Cartesian coordinate system in meters for a certain area in Africa[3]. From that data, we then extract a $W = 401$ meter by $H = 401$ meter area for each bridgelet. As each field in the array provided to the framework represents the land type for a field of both 1 meter in length and width, the area of $W$ by $H$ meters squared can be covered by $[W/10]$ by $[H/10]$ fields in the land cover dataset.

For the experiments with temporal data two temporal mixed random walks have been generated with $T = 200$ time steps and $D_{max} = 1$ (Figure 4.13). In Figure 4.13a random walks

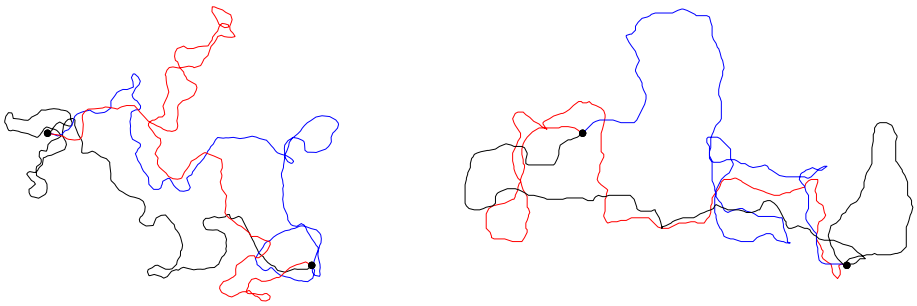
[3] https://epsg.io/32736



(a) Movement models alternate from a biased random motion with a west bias, to Brownian motion, to the same biased random motion.

(b) Movement models alternate from a biased random motion with a west bias, to Brownian motion, to the biased random motion with an east bias.

Figure 4.13: Example of mixed random walks on temporal data.



(a) Correlated random walks generated with the RTG algorithm.

(b) Correlated random walks generated with our method.

Figure 4.14: Comparison of correlated random walks with $T = 300$ generated with the RTG algorithm [83] and our method.

are shown where transition matrix $M_{d,t}$ is a biased transition matrix with a west bias for $0 \leq t \leq 25$ and $175 \leq t \leq 200$, and where $M_{d,t'}$ is a Brownian motion transition matrix for $25 < t' < 175$. Figure 4.13b shows random walks with similar temporal data, except that transition matrix $M_{d,t}$ is a biased transition matrix with an east bias for $175 \leq t \leq 200$.

## 4.2 Analyzing Behavioral Patterns of Small Terrestrial Mammals

To demonstrate the methods described in this chapter, in this section we use the methods described above to analyze the behavioral patterns of a small terrestrial mammal. Gardiner, Hamer, Leos-Barajas, Peñaherrera-Palma, Jones, and Johnson [34] studies how the Eastern Bettong (*Bettongia gaimardi*) relies on habitat indicators to make decisions on whether it will start denning, forage or fast-travel. One of the key findings of their paper is that when the bettong is in the fast-traveling state, it tends to use movement

corridors with relatively high canopy cover, presumably because these corridors provide concealment from predators and reduce exposure while moving between foraging patches and refuge sites. This state-dependent habitat selection motivates our analysis: we investigate whether canopy cover alone contains a detectable signal that differentiates behavioral states, and whether this signal is consistent across individuals and seasons.

### 4.2.1 Data description

The dataset used in this experiment contains GPS location measurements of 26 Eastern Bettongs that were tracked for multiple weeks at three sites in the Midlands bioregion of Tasmania, Australia. Locations were recorded at 15-minute intervals using GPS collars. Each location measurement is tagged with a behavioral state indicating whether the animal was denning, foraging, or fast-traveling at that time. In addition, the dataset contains habitat covariates extracted at each location, including the *Normalized Difference Vegetation Index (NDVI)* and *Canopy Cover (CC)*.

In this section we focus on canopy cover. The canopy cover layer provides an estimate of persistent green vegetation cover at seasonal temporal resolution. Conceptually, it represents the fraction of vegetation that remains persistently green year-round, and therefore serves as a proxy for structural cover and concealment that may influence movement decisions. The spatial resolution of the raster layer is 30 m. Although the animal locations are sampled every 15 minutes (i.e., at a much finer temporal resolution than the habitat layer), the seasonal aggregation is appropriate for our purposes: we treat canopy cover as a slowly varying contextual covariate and align each GPS fix with the canopy-cover season in which it occurs [27]. For each GPS fix, the canopy cover value is obtained by sampling the raster at the recorded coordinates. This yields a univariate time series of canopy cover values per individual, paired with the corresponding behavioral state label.

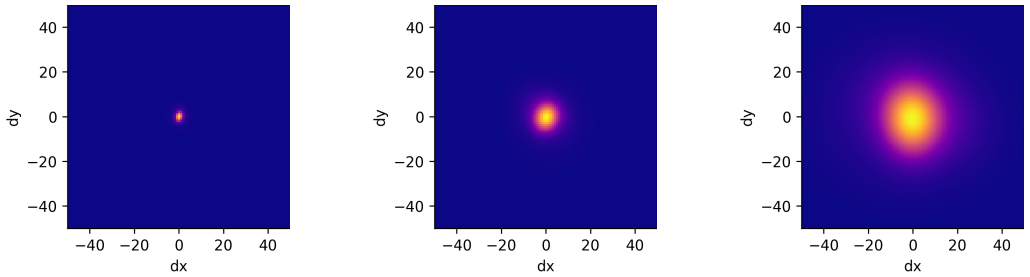
### 4.2.2 Computing transition matrices

As mentioned before, each GPS fix is tagged with a behavioral state indicating whether the bettong was denning, foraging, or fast-traveling at that time. For each state, we want to construct a transition matrix $M_{\text{den}}$, $M_{\text{for}}$, and $M_{\text{ft}}$ that captures the corresponding step distribution.

To do so, we first split the trajectory data by behavioral state and then compute displacement vectors between consecutive fixes whenever the time gap is exactly 15 minutes. These displacement vectors form the empirical 15-minute step samples for that state.

We then fit the 15-minute step distributions using the sum-of-Gaussians approach described in Section 4.1.5. Since the experiments in this section use an uncorrelated random walk model, we rescale these distributions to obtain an equivalent 1-minute step distribution. This rescaling is also convenient because the tracking data are not perfectly regular: while the nominal sampling interval is 15 minutes, occasional gaps of 14, 16, or 30 minutes occur. Working with a 1-minute transition model allows us to accommodate such inconsistencies by simply adjusting the number of time steps.

Next, we choose a step size $R$ for rasterizing the 1-minute step distributions. We select $R$ based on the distribution mass: for each behavioral state, we choose $R$ such that at least 99% of the probability mass lies within the square $[-R, R]^2$. For denning, foraging, and




(a) Transition matrix $M_{\text{den}}$ for denning behavior.

(b) Transition matrix $M_{\text{for}}$ for foraging behavior.

(c) Transition matrix $M_{\text{ft}}$ for fast-traveling behavior.

Figure 4.15: Transition matrices for the three different behavioral states of the Eastern Bettong.

fast-traveling we use $R = 50$. Finally, following Section 4.1.5, we rasterize the 1-minute step distributions to obtain the transition matrices $M_{\text{den}}$, $M_{\text{for}}$, and $M_{\text{ft}}$ (Figure 4.15).

### 4.2.3 Calculating utilization distributions

The central object in these experiments is the utilization distribution between two location measurements, as introduced in Section 4.1.2. We use the UD in two ways.

First, it provides an estimate of the space use associated with an individual movement step, rather than treating the animal as having occupied only the observed GPS fixes. A common approach to summarizing space use is the home range, which is often defined via a UD isopleth. Following Börger, Franconi, De Michele, Gantz, Meschi, Manica, Llovari, and Coulson [16], we estimate the home range HR as the 90% isopleth of the UD, i.e., the smallest area that contains 90% of the total utilization probability mass. More concretely, we compute a UD for each step, extract its 90% isopleth, and then take the union of these step-wise isopleths to obtain a home range estimate for the full tracking period (Figure 4.16). Using the home range, we can calculate what fraction of the home range falls into different canopy cover classes (e.g., low, high). This provides a baseline expectation of habitat availability for the individual during the tracking period:

$$A(c) = \frac{|\{z \in \text{HR} : z \in \mathcal{Z}_c\}|}{|\text{HR}|},$$

i.e., the fraction of home-range area in canopy class $c$.

Secondly, we use the UD to analyze habitat selection during movement. For each step, we compute the UD and then extract the canopy cover values at all cells with non-zero utilization probability. This yields a weighted sample of canopy cover values that reflects the habitat use during that step. By aggregating these samples across steps and individuals, we can compare habitat use between behavioral states.

$$U_s(c) = \frac{\sum_{z \in \mathcal{Z}_c} \text{UD}_s(z)}{\sum_{z \in \mathcal{Z}} \text{UD}_s(z)},$$

where $\mathcal{Z}_c$ denotes the set of grid cells belonging to class $c$ and $\mathcal{Z}$ the full local grid. The quantity $U_s(c)$ therefore represents the fraction of step-wise UD probability mass falling into canopy class $c$.

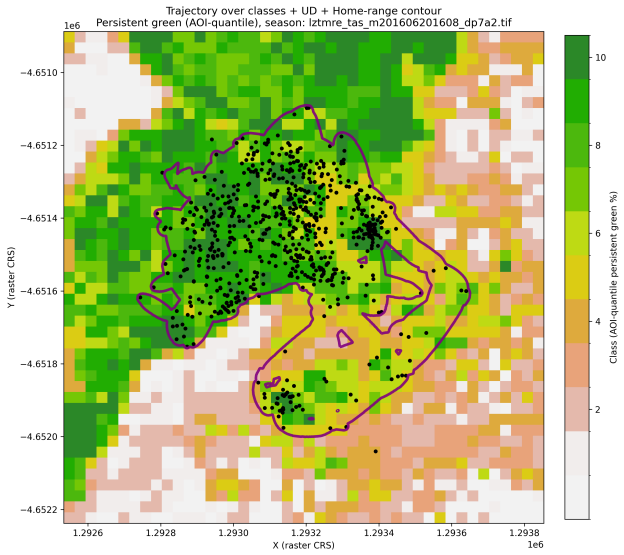



Figure 4.16: Home range (purple) and location measurements (black) of the Eastern Bettong called *Sifa*. The background color indicates canopy cover, with greener colors representing higher canopy cover. The home range is estimated as the union of step-wise 90% UD isopleths, which are computed using the transition matrices corresponding to the behavioral state at each step.

### 4.2.4 Habitat selection analysis

We test whether canopy cover contains a state-dependent signal by comparing canopy use during each movement step to a baseline of habitat availability defined by the individual's home range.

**Selection scores.** For each individual $i$, we define the home range $\mathrm{HR}_i$ as the union of step-wise 90% UD isopleths (Section 4.1.2). We then compute the mean canopy cover available within the home range by sampling the seasonal canopy raster on all valid raster cells inside $\mathrm{HR}_i$:

$$\mu_i^{\mathrm{HR}} = \frac{1}{|\mathrm{HR}_i|} \sum_{z \in \mathrm{HR}_i} C(z),$$

where $C(z)$ is the canopy cover percentage at grid cell $z$ and $|\mathrm{HR}_i|$ denotes the number of valid home-range cells.

**Step-wise canopy use.** For each movement step $s$ of individual $i$, we compute the step utilization distribution $\mathrm{UD}_s(z)$ on a local grid and extract the canopy cover at all grid cells with non-zero utilization probability. We summarize canopy use during the step by the UD-weighted mean canopy cover,

$$\mu_s^{\mathrm{UD}} = \frac{\sum_z \mathrm{UD}_s(z)\, C(z)}{\sum_z \mathrm{UD}_s(z)}.$$

Table 4.1: State contrasts in canopy-cover association relative to home-range availability. Reported values are means of paired within-individual contrasts in $\Delta^{UD}$, with 95% bootstrap confidence intervals. $k_+$ is the number of individuals with a positive paired contrast, and $p_+ = k_+/n$.

| Sex | Contrast | Mean | 95% CI | $n$ | $k_+$ ($p_+$) |
|---|---|---|---|---|---|
| male | foraging–fast | 1.44 | [0.48, 2.59] | 14 | 12 (0.86) |
| male | foraging–denning | 0.60 | [-0.48, 1.53] | 14 | 11 (0.79) |
| male | fast–denning | -0.84 | [-2.56, 0.67] | 14 | 6 (0.43) |
| female | foraging–fast | 0.77 | [-0.46, 2.00] | 11 | 8 (0.73) |
| female | foraging–denning | 0.83 | [-0.66, 2.45] | 11 | 7 (0.64) |
| female | fast–denning | 0.06 | [-2.13, 2.52] | 11 | 4 (0.36) |

**Selection indices.** We quantify selection relative to home-range availability using simple differences,

$$\Delta_s^{UD} = \mu_s^{UD} - \mu_i^{HR}.$$

Positive values indicate that the step is associated with higher canopy cover than the individual's home-range average, while negative values indicate use of relatively lower canopy cover.

**State aggregation and contrasts.** Each step is labeled by its behavioral state at the start of the step (denning, foraging, fast-traveling). To avoid overweighting individuals with more steps, we first average the step-wise indices within each individual $i$ and state $b$, yielding $\overline{\Delta}_{i,b}$. We then compute paired within-individual contrasts such as

$$\Delta^{(i)}_{\text{for-fast}} = \overline{\Delta}_{i,\text{foraging}} - \overline{\Delta}_{i,\text{fast}}, \qquad \Delta^{(i)}_{\text{for-den}} = \overline{\Delta}_{i,\text{foraging}} - \overline{\Delta}_{i,\text{denning}},$$
$$\Delta^{(i)}_{\text{fast-den}} = \overline{\Delta}_{i,\text{fast}} - \overline{\Delta}_{i,\text{denning}}.$$

These contrast values directly test whether canopy association differs between behavioral states. All analyses are stratified by sex by splitting individuals into male and female groups. Individuals that never enter one of the compared states do not contribute to the corresponding paired contrast.

**Uncertainty estimation.** We estimate uncertainty in mean state contrasts using nonparametric bootstrap resampling over individuals within each sex. For each bootstrap replicate we resample individuals with replacement, recompute the mean contrast, and report the resulting 95% bootstrap confidence interval. In addition, we report the proportion of individuals for which the paired contrast is positive as a simple directional consistency check.

### 4.2.5 Results

Using the home-range mean canopy cover as the availability baseline, we summarize step-wise canopy association by the difference $\Delta^{UD}$ between the UD-weighted mean canopy cover during a step and the individual home-range mean. Positive values indicate steps occurring in relatively higher canopy cover than is typical within the individual's home range.

Table 4.1 summarizes paired within-individual state contrasts in $\Delta^{UD}$, stratified by sex, along with 95% bootstrap confidence intervals and a directional consistency check. In males, the foraging-fast contrast is the most clearly separated from zero: the mean paired difference is positive and the bootstrap interval excludes zero, with most males showing a positive paired contrast. In females, the corresponding foraging-fast contrast is also positive on average but more variable across individuals, and its confidence interval overlaps zero. Contrasts involving denning (foraging-denning and fast-denning) are more uncertain in both sexes, with wider intervals overlapping zero and weaker directional consistency. However, in males the fast-denning contrast is negative on average, indicating that fast-travel steps tend to be associated with lower canopy cover than denning steps, even though the bootstrap interval overlaps zero.

# Chapter 5
# Two-Point Link Distance Queries in Polygonal Domains

In this chapter we study minimum-link paths within the context of polygonal domains. A minimum-link path between points $s$, $t$ in a polygonal domain $P$ is a polygonal $s$-$t$ path with the minimum number of edges (links). The number of links in a minimum-link path is the *link distance* between $s$ and $t$. Furthermore, the *link diameter* of $P$ is the maximum possible link distance between any two points in the domain. The *link center* of the domain is the point minimizing the maximum link distance to points of $P$; this maximum distance is the *link radius*. The above notions are analogous to the same concepts (diameter, center, radius) for arbitrary metric spaces.

The minimum-link path problem is a natural geometric optimization problem which is closely related to the well-studied problem of computing geodesic (shortest) paths in polygonal domains. However, both geodesic (shortest) and minimum-link paths may be more intricate than they may appear at first glance:

- It is folklore that geodesic paths may be found by searching the visibility graph of the domain. However, even when all vertices have integer coordinates, a subtle computational difficulty arises when comparing path lengths. The length of any candidate path is the sum of Euclidean distances of its segments, and each such distance is the square root of an integer. Consequently, the total path length is a sum of square roots. Although this quantity is well defined mathematically, no polynomial-time algorithm is known for deciding whether such a sum is less than, equal to, or greater than a given integer. Thus, even verifying whether a candidate path has length at most a specified bound is not known to be solvable in polynomial time.
- As explained in Section 5.1.1, vertices of a minimum-link path cannot always be snapped to a discrete set of points, leading to high bit complexity of the paths even in simple polygons [49, 54]. In a polygonal domain with holes, it is not immediately clear how to find a minimum-link path in polynomial time. One may compute the $k$-link reachable regions, from the starting point, for increasing $k$, but bounding the complexity of the regions (and hence the algorithm's runtime) requires an insight [63].

The problems considered in this chapter (2-point shortest path queries, diameter and radius) are substantially more challenging. Quoting [6]: "no algorithm for computing the diameter and radius under the link distance is known, not even one that runs in exponential time." We show that the problems admit polynomial-time solutions. The runtimes are high: improving them is left for future work.

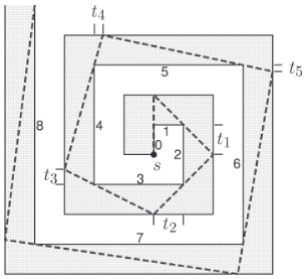



Figure 5.1: Figure 1 from [49]: the coordinates of the vertex $t_1$ of the path are obtained as a solution to a system of 2 linear equations with 2 unknowns and integer coefficients. Thus, these coordinates are represented by a ratio of integers; subsequent bend points are solutions to systems of linear equations whose coefficients have higher and higher bit complexity (they are ratios of larger and larger integers).

## 5.1 Preliminaries and related work

Let $n$ denote the number of vertices of polygonal domain $P$. The *visibility polygon* (VP) of a point $p \in P$ is the set of points in $P$ seen by $p$. We say that a point $q \in P$ is *seen by* a point $p \in P$ if the line segment $pq$ lies entirely in $P$ The *weak visibility polygon* of a subset $S \subseteq P$ is the union of the VPs of the points of $S$; equivalently, the weak visibility polygon is the set of points seen by at least one point of $S$. The *visibility graph* (VG) of $P$ is the graph on vertices of $P$, whose edges connect pairs of mutually visible vertices. We assume that edges of $P$ are also edges of the VG (i.e., that neighboring vertices of $P$ see each other along the boundary edge).

### 5.1.1 Link distance and bit complexity

Computing the link diameter and radius/center of a polygonal domain with holes has been open since analogous problems were considered for simple polygons over 30 years ago [80, 5]. The reason why even an exponential-time solution is not obvious is because vertices of a minimum-link path do not necessarily lie on the VG or some other discrete structure determined from the polygon (Figure 5.1): it was shown [49, 54] that $\Theta(n \log n)$ bits may be required to represent coordinates of vertices of the minimum-link path, even if the vertices of $P$ have integer coordinates specified with $O(\log n)$ bits.

### 5.1.2 Staged illumination

Algorithms for computing the link distance from a source point $s$ [78, 40, 44, 63] employ the "staged illumination" paradigm (see, e.g., the handbooks [71, Chapter 12] and [85, Chapter 31.3]): At the first stage, place the light source at $s$ and illuminate VP of $s$ which is the set of points with link distance 1 from $s$. At the beginning of any subsequent stage, the boundary between the lit and the dark portions of $P$ is defined by a set of line segments we call *windows* (any window starts at a vertex and ends on the boundary of $P$) which bound the weak visibility polygon of the area lit at the previous stage.

Figure 5.2 (left) illustrates the staged illumination in a simple polygon. To efficiently run the staged illumination in a polygonal domain with holes [63], some chains of windows

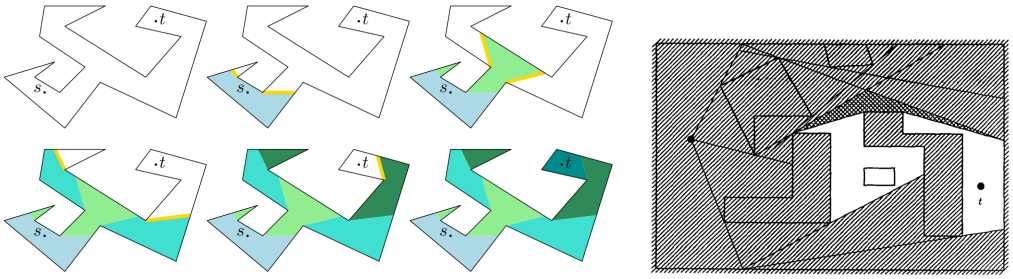


Figure 5.2: Staged illumination in a simple polygon (left) and in a polygonal domain with holes (right). Left: Figure 3 from [62]: The windows are yellow. Right: Figure 5 from [63]: The cross-hatched area is what is swept by the lower envelope of a set of windows when this lower envelope is a "rubber band" pinned at its endpoints: when released, the rubber band snaps onto edges of the VG.

are replaced by their relative convex hulls within $P$ Ṫhis replacement essentially snaps the windows to the edges of the VG. See Figure 5.2 (right) and [63], in particular, Figures 2-5 therein for the details.

The *greedy* path [63, 5] from $s$ to a window $w$ is the minimum-link path whose links are all aligned with windows and whose last link is aligned with $w$. E.g., the path in Figure 5.1 is greedy.

### 5.1.3 Distance map

The *link distance map*, denoted LDM($s$), from the *source* point $s$ is a decomposition of $P$ into cells such that the link distance from $s$ to any point within one cell is the same. Note that it is not required that the cells of an LDM are maximal (i.e., that the link distance necessarily changes as you step from a cell to a neighboring cell): the only requirement is that the distance never changes within a cell.

For simple polygons, an LDM is really a by-product of the staged illumination (the edges of the map are the windows). Both a single minimum-link path and the LDM can be computed in $O(n)$ time (because in a simple polygon the windows are pairwise disjoint, the LDM in it is also called a "window partition" [80, 5]).

In polygons with holes the LDMs can have $\Omega(n^4)$ complexity [81], and extra care must be taken in order to produce the LDM. In [63, Theorem 12] it was shown how to build the LDM in $\tilde{O}(n^5)$ time, where $\tilde{O}$ notation suppresses polylogarithmic factors. The map is defined by an arrangement of windows (called "illumination edges" in [63]) within the domain.

Similarly to constructing an LDM from a point, one can build an LDM from a source *chord* $c$ of $P$ (Figure 5.3): the weak VP of $c$ is the set of points reachable with 1 link from $c$, and the subsequent steps of the illumination are defined in the same way as for a point source. LDMs from segments were built already in early work [80] on link distance; in particular, LDMs from chords were used for finding the link center of a simple polygon [28].

### 5.1.4 1-point queries

After being preprocessed for point location, an LDM allows one to report the link distance from the source to a query point $q$ in optimal $O(\log n)$ time, by locating the cell of the

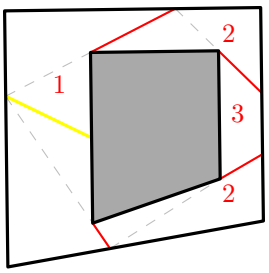



Figure 5.3: LDM($c$) is obtained by starting the illumination from $c$ (yellow); the edges of the map are solid red, and the cells are marked with the link distance to $c$.

map that contains $q$ (enhanced with the backpointers, the map lets one report an actual minimum-link path in the additional time proportional to the number of links). An LDM is thus analogous to the *shortest-path map* (SPM($s$)) in $P$, which decomposes the domain into cells such that the length of the geodesic (shortest) path from $s$ to any point in one cell is given by the same formula. Note that while SPMs have linear complexity and can be built in linear (for simple polygons) or near-linear (for polygons with holes) time [95, 95], finding even a single minimum-link path in polygons with holes is 3SUM-hard [62] (an $\tilde{O}(n^2)$-time algorithm is given in [63]).

## 5.2 Prior work on 2-point queries and our results

While LDMs and SPMs give complete answers to the *1-point* link and geodesic distance query respectively (for a *fixed* source, report the distance from a query point to the source), for the *2-point* distance query problem (report the distance between two query points $s$ and $t$), the solutions achieving optimal $O(\log n)$ query time are much more involved [93, 24, 42, 5, 11]:

- An optimal (linear-preprocessing) algorithm is known only for 2-point geodesic distance queries in simple polygons [42].
- A cubic-size data structure for link distance queries in simple polygons was given in [5].
- For geodesic distance queries in polygonal domains, two methods were presented in [24]:
  - **2d SPM equivalence decomposition** splits $P$ into a polynomial number of cells such that SPM($s$) remains combinatorially the same (called "topologically equivalent" in [24, 7]) while $s$ stays within one cell, and stores a *parametric* SPM for each cell (the parameter being the location of $s$ within the cell: the parametric SPM records how the edges of SPM($s$) change depending on where $s$ is). After preprocessing the parametric SPM for parametric point location [24, Section 4.2], 2-point geodesic distance queries may be answered by locating $s$ in the decomposition of $P$ and then locating $t$ in the parametric SPM($s$).
  - **4d geodesic distance equivalence decomposition** splits $P \times P$ into (a polynomial number of) cells such that the geodesic distance between any pair of points ($s$, $t$) within one cell is given by the same formula, i.e., by the same function of ($s$, $t$). (On a detailed note, [24] presented two 4d structures: the

vanilla $O(n^{22+\varepsilon})$-space 4d decomposition introduced in [24, Section 3] first, and the improved $O(n^{15+\varepsilon})$-size data structure in [24, Theorem 3.1]. We refer to the former decomposition because the improved one, being specific to geodesic paths, does not work for us; similarly we did not see how to extend to link distance the recent data structure for geodesic paths from [11].)

- No data structure for 2-point link distance queries in polygonal domains with holes has been known previously. One of the contributions of this chapter is working out such data structures (analogous to the 2d and the 4d decomposition for geodesic queries [24]) by extending the solution of [5] (for 2-point link distance queries in simple polygons) to polygons with holes. Specifically, we provide the following two data structures:
  - **2d LDM equivalence decomposition** is the decomposition of $P$ into (a polynomial number of) cells such that LDM($s$) remains combinatorially the same while $s$ stays in one cell. The map, i.e., its vertices and edges (windows), is a known function of $s$ (the functions may be different in different cells, but within one cell the function is same).
  - **4d link distance equivalence decomposition** decomposes $P \times P$ into (a polynomial number of) cells such that the link distance between any pair $(s, t)$ in one cell is the same.

### 5.2.1 Diameter and radius/center

Computing the diameter and radius is intimately connected to the 2-point distance query structures, described above, via the following observation applicable to both geodesic and link distances

**MetaTheorem 26.** *If the 2d map equivalence decomposition of $P$ or the 4d distance equivalence decomposition of $P \times P$ can be built in polynomial time, then the diameter and the radius/center of $P$ can be computed in polynomial time.*

*Proof.* Go through every cell $\sigma$ of the 2d map equivalence decomposition.

- For the geodesic distance, the maximum distance from $s$ is attained at a vertex of SPM($s$) [7, Lemma 1]. Since in every cell $\sigma$, it is known how SPM($s$) changes with $s$, distances from $s$ to all vertices are also known, and hence the maximum distance from $s \in \sigma$ to a point in $P$ is also known (from the upper envelope of the distances).
- For the link distance, the maximum distance from $s \in \sigma$ is the distance to points in the cell of LDM($s$) farthest from $s$ (any point in a cell of LDM($s$) has the same link distance from $s$ thus there is no need for the envelope).

To identify the diameter, take the maximum distance to the furthest point, and for the radius take the minmax; for the final answer, choose the best $\sigma$ accordingly. To find the diameter, one may also go through every cell of the 4d decomposition and in every cell find the pair $(s, t)$ that maximizes the $s$-$t$ distance. Finally, the diameter will be given by the overall maximum. □

| | simple polygon | | polygonal domain | |
|---|---|---|---|---|
| | Diameter | Radius | Diameter | Radius |
| Geodesic | $O(n)$ [45] | $O(n)$ [3] | Poly-time [8] | Poly-time[7, 94] |
| Link | $O(n \log n)$ [79] | $O(n \log n)$ [28, 52] | Poly-time [this chapter] | |

Table 5.1: A sub-table of [6, Table 1]: known results for computing the diameter and radius in simple polygons (left) and polygonal domains (right)

### 5.2.2 State of the art and our contribution

Solutions based on MetaTheorem 26 are not very efficient, and faster algorithms exist for finding both the geodesic diameter [8] and radius [94] of polygonal domains (in simple polygons, linear-time algorithms exist for both geodesic diameter [45] and radius [3]; link diameter [79] and radius [28] of simple polygons can be found in near-linear time). Existing results regarding diameter and radius are summarized in Table 5.1: essentially, the only case that has been missing is link distance in polygons with holes. Similarly, for simple polygons 2-point distance query data structures are known both for geodesic [42] and link distance [5], while for polygonal domains the problem has been solved only for the geodesic distance [24].

In this chapter, we fill the gaps by showing how to construct the 2d map equivalence decomposition of a polygonal domain $P$ and the 4d distance equivalence decomposition of $P \times P$ under the link distance. By MetaTheorem 26, this leads to polynomial-time algorithms for finding the link diameter and radius/center of the domains $P$.

The rest of the chapter is organized as follows. Section 5.3 gives a simple algorithm to compute the link diameter (without resorting to MetaTheorem 26), and Section 5.4 presents data structures for 2-point link distance queries. Our data structures reuse, extend and combine the techniques of [5] (link distance queries in simple polygons) and [24] (geodesic distance queries in polygonal domains with holes): analogously to [24], our data structures are the 2d LDM equivalence decomposition and the 4d link distance equivalence decomposition.

Throughout this chapter, "distance" will mean link distance, "length" of an $s$-$t$ path will mean the link distance between $s$ and $t$ along the path, etc. Slightly abusing the terminology, we will use the term *diameter* to mean both a minimum-link path between two diametrical endpoints and the number $d$ of links in such a path.

## 5.3 A simple algorithm for link diameter: the details

### 5.3.1 Diametrical endpoint at a vertex

One difficulty in finding diameters (both link and geodesic) in domains with holes is that the diameter may be defined by points in the interior of $P$ (Figure 5.4). The opposite, easiest, case is when there exists a diameter with an endpoint on a vertex of $P$; in this case, the diameter can be found by building LDMs from every vertex of $P$ The diameter is then simply the maximum distance from a vertex of $P$ to a point of $P$, which can be read off from the LDMs. In what follows we will assume that this diameter candidate (with an endpoint on a vertex) has been computed, and we will show how to find the diameter under the assumption that there exists no diameter with an endpoint lying on a vertex of the domain.

### 5.3.2 Diameter 1 or 2

Trivially, the diameter is 1 if and only if $P$ is a convex polygon. The diameter is at most 2 if and only if for any $s, t \in P$ there is a point $p \in P$ visible from both $s$ and $t$: then $p$ is the bend of a 2-link $s$-$t$ path.

We now use the following simple fact: if a link of a minimum-link path is not supported by a vertex of $P$, then we may rotate it continuously, preserving feasibility and without increasing the number of links, until it is supported by a vertex. Thus, if $s$ and $t$ admit a 2-link path, then there are vertices $v$ and $u$ of $P$, visible to $s$ and $t$, respectively, such that the rays $sv$ and $tu$ intersect, at a point $p$, before exiting $P$.

We look at the question from the "other" side, i.e., from the point of view of $v$ and $u$: we go through every pair of vertices $v, u$ and compute the set $P_{vu} \subseteq P \times P$ of pairs $(s, t)$ such that $v$ and $u$ may support the links of a 2-link $s$-$t$ path. If the union of the sets $P_{vu}$, for all pairs $v, u$, fully covers $P \times P$, then $d \leq 2$; otherwise, there exist points $s, t$ with link distance at least 3.

To compute $P_{vu}$, we first decompose $P$ by extensions of VG edges: any point $p$ within one cell of the decomposition sees the same set of vertices; in particular, we consider only the cells from which both $v$ and $u$ are seen. As $p$ moves within such a cell $\sigma$, the rays $pv, pu$ rotate around $v, u$, respectively, sweeping the locations for $s, t$, respectively, and thus creating $P^{\sigma}_{vu} \subseteq P_{vu}$ i.e., the subset of $P_{vu}$ corresponding to having $p$ in $\sigma$. To implement the creation of $P^{\sigma}_{vu}$, triangulate $\sigma$: the subset of $P^{\sigma}_{vu}$ obtained from $p$ in one triangle has constant description complexity, and $P^{\sigma}_{vu}$ is the union of such subsets over all the triangles.

### 5.3.3 Endpoints in the interior and $d \geq 3$

Finally we address the most general question of finding a long diameter when no vertex is a diameter endpoint. Our main structural observation, which leads to a simple algorithm for finding the diameter (running in high but polynomial time), is the following:

**Observation 27.** *If $d > 2$ and no diameter has an endpoint on a vertex of $P$, then some diameter can be modified so that one of its edges is aligned with a VG edge.*

*Proof.* Let $\pi$ be a diametrical $s$-$t$ path with $d$ links. W.l.o.g. we may assume that the first

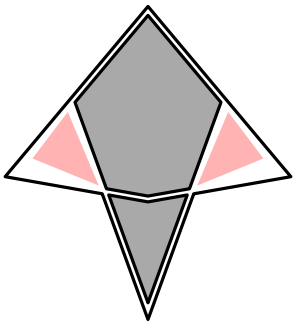


Figure 5.4: Holes are gray. Both endpoints of any link diameter lie in the red regions, i.e., in the interior of $P$ (cf. similar examples for the rectilinear case [6, Fig. 1] and geodesic diameter [8, Fig. 1(c)]).

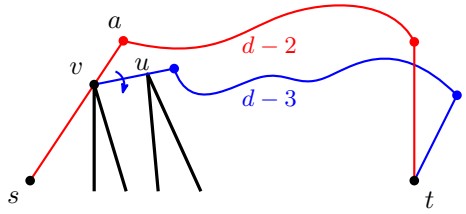



Figure 5.5: Aligning the second edge of diameter with a VG edge.

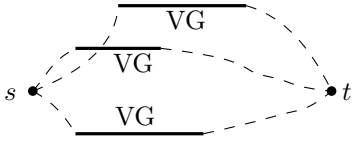



Figure 5.6: VG-restricted distance is along a minimum-link path that uses a link aligned with an edge of VG.

link of $\pi$ is supported by a vertex of $P$ (if it is not, rotate the link until it is). Let $v$ be the vertex supporting the first link, say $sa$ (Figure 5.5). Then, we can reason about the distance from $v$ to $t$.

- It cannot be more than $d$, because starting from $v$ we may follow the segment $va$ and then the remaining part of the diametrical path from $a$ to $t$, obtaining a $v$-$t$ path with at most $d$ links.
- If it is $d$, then there is a diameter with an endpoint at a vertex ($v$), contrary to the assumption.
- It cannot be smaller than $d-1$, or else the path $s$-$v$-$t$ would have fewer than $d$ links.

Thus the distance from $v$ to $t$ is exactly $d-1$. Then the path $s$-$v$-$t$ is a diameter: it connects two diametrical endpoints $s$ and $t$, and has length $d$. Now take a shortest $v$-$t$ path. Its first link can be rotated until it hits a vertex $u$ visible to $v$; this way the link becomes aligned with the edge $vu$ of VG, and the resulting $s$-$t$ path is still diametrical. □

### 5.3.4 Finding long diameter with no vertex endpoint

Define the *VG-restricted* link distance between $s, t \in P$ as the minimum, over all VG edges $e$, of distance from $s$ to the maximal extension of $e$, plus 1 (for a link aligned with $e$), plus distance from the extension to $t$; the VG-restricted diameter is the maximum VG-restricted distance between points in $P$ (Figure 5.6). By Observation 27, if $d > 2$ and no vertex is a diameter endpoint, the VG-restricted diameter is equal to the true diameter.

To compute the VG-restricted diameter, build the overlay of LDMs from maximal extensions of all visibility edges (recall that LDM from a chord is computed by staged illumination in the same way as from a point). By construction, the distance to any such extension is the same for all points in one cell of the subdivision, and hence the VG-restricted distance between any two points $s \in \sigma, t \in \tau$ within two fixed cells $\sigma, \tau$ of the subdivision is the same. In particular, to find the diameter, it suffices to compute the VG-restricted distance between two arbitrary points in every pair of cells (this is just a shortest-path computation in a graph).

The final value of the diameter is the maximum of the value computed here and the value

computed under the assumption that there is a diameter with an endpoint on a vertex (computed in Section 5.3.1); if this maximum is at most 2, then the preceding subsection already handles this case.

## 5.4 Two-point link distance queries

In this section we show how to compute the 2d LDM equivalence decomposition and the 4d link distance equivalence decomposition. These yield data structures for the 2-point link distance queries, analogous to the structures of [24] for geodesic queries (see Section 5.2). Our data structures extend the data structure of [5] (for 2-point link distance queries in simple polygons) to polygonal domains with holes. The extension, allowing us to track combinatorial changes in LDM($s$) for $s \in P$, is two-fold:

**Same windows.** In addition to decomposing the boundary of the polygon into "atomic segments" (as was done in [5]), we decompose the entire $P$ into "atomic cells" by overlaying LDMs from extensions of VG edges. For every $s$ in one cell $\sigma$ of the decomposition, LDM($s$) has the same set $W(\sigma)$ of windows. Moreover, the same subset $R(\sigma) \subseteq W(\sigma)$ of windows rotates as $s$ moves within $\sigma$, while the other windows, $W(\sigma) \setminus R(\sigma)$, are not influenced by the location of $s$ in the cell.

**Same arrangement of windows.** We track all possible intersections among windows in LDM($s$), which may change because the rotating windows sweep through the domain as $s$ moves.

### 5.4.1 Extended visibility graph

For a line segment $\ell$ in $P$ let $\bar{\ell}$ be the chord of $P$ containing $\ell$ (i.e., $\bar{\ell}$ is $\ell$ extended maximally within $P$). Define the Extended Visibility Graph EVG = $\{\overline{uv} : uv$ is an edge of the VG$\}$ as the set of chords of $P$ obtained by maximally extending every edge of the VG (we remind that edges of $P$ are also edges of the VG).

### 5.4.2 Re[de]fining LDM

Recall [63, Section 8] (see also Section 5.1.3) that an LDM is defined by an arrangement of windows together with edges of $P$. Recall also that there is no requirement that the cells of an LDM are maximal (i.e., that the link distance necessarily changes as you move between cells). The only requirement of the LDM is that the distance never changes within a cell. Thus LDM($s$), for some point $s \in P$, remains LDM($s$) even if one subdivides its cells into smaller pieces.

For the rest of the chapter, we redefine LDMs to include all chords of the EVG. That is, we refine an LDM by chords of the EVG (but we continue to call the result LDM). In other words, we assume that the EVG chords are windows of LDM($s$) for any $s$: we thus use the term "windows" to refer both to "original" windows of the LDM (as defined in [63]) and chords of the EVG.

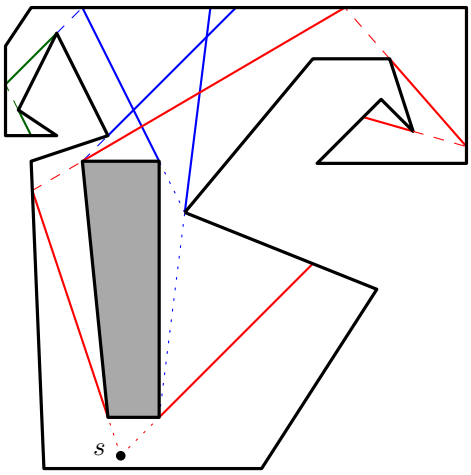



Figure 5.7: Pinned windows (those not aligned with edges of $P$) are blue (the aligned VG edges are dotted blue), static bash windows are green, rotating windows are red. For clarity, some windows have been omitted from the image.

### 5.4.3 Static and rotating windows

Every window $w$ necessarily goes through a vertex of $P$ (Figure 5.7). If the chord $\overline{w}$ goes through another vertex ($\overline{w} \in$ EVG), we call $w$ a *pinned window*. Such windows do not change under small local motions of $s$; in particular, chords of the EVG are pinned windows. Otherwise we call $w$ a *bash window*, and the endpoints of $\overline{w}$ the *bash points*. A bash point lies in the interior of an edge of $P$; we call such an edge a *bash wall*. (The term "pinned" is borrowed from [5] where analogous edges were called "(rotationally) pinned"; the term "bash" is borrowed from [61] where path vertices lying in the interior of polygon edges were called "bash points" and the edges were called "bash walls".)

As $s$ moves slightly, pinned windows of LDM($s$) do not change. A bash window $w$ may remain still or may rotate, depending on whether there is a pinned window appearing on the greedy path from $s$ to $w$ (recall from Section 5.1.2 that a greedy path is the minimum-link path following the windows):

**Static bash windows.** Any bash window $w$ appearing after a pinned window $w'$ does not move with $s$ because $w$ is a window in LDM($\overline{w'}$) and $w'$ does not move with $s$.

**Rotating bash windows (bash-bash windows).** If all windows preceding $w$ are bash windows, then they all rotate around vertices of $P$, with the bash points (which are the bend points of the greedy path from $s$ to $w$) sliding along the bash walls; we call such windows *bash-bash windows* and the greedy path from $s$ to $w$ a *bash path*. (See Figure 5.7: as $s$ moves left, all windows rotate, so they are bash-bash windows. This bashing is what leads to the high bit complexity of minimum-link-path vertices and what makes minimum-link paths hard to study, since bash points do not lie on any predefined structure computable from $P$, unlike geodesic paths, which follow the visibility graph.)

Overall, we obtain the classification of LDM windows into static and rotating windows. The former may be EVG chords or static bash windows and the latter are bash-bash windows.

### 5.4.4 LDMs overlay

We build LDMs from all chords of the EVG; let $W_{\text{EVG}}$ denote the set of all windows in these LDMs. The first step in constructing our data structures is computing the overlay $\mathcal{O}$ of the windows in $W_{\text{EVG}}$. That is, we compute the overlay of LDMs from all chords of the EVG. This overlay is the same construction as [5] used, but we build the full overlay of the LDMs, while [5] considered only the interaction of $W_{\text{EVG}}$ with the boundary of $P$.

Consider LDM($s$) for a point $s$ in $P$. As $s$ moves, the bash-bash windows of LDM($s$) rotate and the map may change combinatorially when either

**Simple case.** A window hits a vertex of $P$ (aligning itself with a VG edge) when $s$ crosses an edge of $\mathcal{O}$, or

**New case.** The arrangement of the windows changes because three windows pass through a common point (out of the three windows, at least one must be a rotating window).

We called the first event "simple" because this is the only combinatorial change that may happen in a simple polygon (where windows are pairwise disjoint). As we will show in Section 5.4.10, to account for new events one needs to build LDMs from intersection points of windows in $W_{\text{EVG}}$, as well as to do some additional, less straightforward computations described later.

An important connection between the overlay $\mathcal{O}$ and static windows in LDMs is that all possible static windows in an LDM from any point of $P$ are known in advance:

**Observation 28.** *Any static window in any LDM belongs to $W_{\text{EVG}}$ (i.e, $\forall s \in P$, if $w$ is a static window in LDM($s$), then $w \in W_{\text{EVG}}$).*

*Proof.* If $w$ is a chord of the EVG, the observation is trivially true: $w$ is a window in LDM($w$). If $w$ is a static bash window, then $w$ is a window in LDM($\overline{w'}$) for some pinned window $w'$, meaning that $w'$ is aligned with a VG edge, implying that $\overline{w'}$ is a chord of the EVG. □

Note that we do not claim that any window from $W_{\text{EVG}}$ is necessarily a window in every LDM; we only claim that $W_{\text{EVG}}$ is a superset of static windows in every LDM.

### 5.4.5 Atomic segments and projection functions

Edges of $\mathcal{O}$ (windows of $W_{\text{EVG}}$) split the edges of $P$ into segments called *atomic* in [5] (note that vertices of $P$ are endpoints of atomic segments too). Since any atomic segment $a$ fully belongs to a single cell in $\mathcal{O}$, for all points $s \in a$, LDM($s$) has combinatorially the same bash paths. Moreover, the exact location of the bash points (vertices of the paths) and hence the directions of the rotating bash-bash windows in LDM($s$) depend on the location of $s$ in $a$ via "projection" functions worked out in [5] which further refers to [2] for details of computing the functions: any projection function is the ratio of 2 linear functions of $s$, whose parameters can be calculated window-by-window in constant time per window.

### 5.4.6 Parametric maps (for simple cases)

If the new cases are ignored, then LDMs (with the corresponding projection functions) built for all cells of $\mathcal{O}$ define the 2d LDM equivalence decomposition. The data structure

can be used to answer 2-point link distance queries in the same way as the 2d SPM equivalence decomposition [24, Section 4.2] answers geodesic distance queries: given query points $s$ and $t$, first $s$ is located in a cell $\sigma$ of $\mathcal{O}$ and then $t$ is located in the parametric LDM($s$).

In fact, since LDM edges (the windows) are straight line segments (not hyperbolic arcs as edges of an SPM), one possibility for locating $t$ in the parametric LDM($s$) is to use the monotone subdivision method of [29]. Indeed, each comparison needed to answer the point location query for $t$ using [29], can be done in constant time even when the edges of the monotone subdivision $\mathcal{M}$ are constant-algebraic-complexity functions of $s$, as long as $\mathcal{M}$ remains the topologically the same (i.e., as long as the horizontal ordering of its vertices remains fixed). Since we know, via the projection functions, how LDM($s$) changes, we also know how $\mathcal{M}$ changes with $s$. In particular, we can compute the locus $S \in \sigma$ of positions for $s$ at which 2 vertices of $\mathcal{M}$ have the same $x$-coordinates: for any 2 vertices $u, v$ of $\mathcal{M}$, this amounts to solving for $s$ the constant-degree polynomial equation $x_v(s) = x_u(s)$ where $x_u, x_v$ are the abscissae (i.e. the $x$-coordinates) of $u, v$. After refining $\sigma$ by such sets $S$ for all pairs of vertices of $\mathcal{M}$, we obtain the required subdivision in which parametric point location query, and hence the link distance query, can be answered in time logarithmic in the complexity of $\mathcal{M}$. Since the complexity of an LDM and hence the complexities of $\mathcal{O}$ and $\mathcal{M}$ are polynomial in $n$, the query time is $O(\log n)$.

### 5.4.7 4d link distance equivalence decomposition (for simple cases)

We can also build the decomposition $\mathcal{O} \times \mathcal{O}$ of the 4d $P \times P$ space. That is, for any point $(s,t) \in P^2$ within one cell $\rho$ of $\mathcal{O}^2$, we have $s \in \sigma, t \in \tau$ where $\sigma, \tau$ are some cells of $\mathcal{O}$. Via the projection functions, we know how each window $w$ of LDM($s$) depends on $s$. In 4d, the union $\bigcup_{s \in \sigma} w(s)$ of the windows $w$ as $s$ varies over $\sigma$ defines a surface of constant algebraic degree, which we call a *curtain* (the surface is swept by the rotating $w$: as $s$ moves along a ray from a vertex $v$ of $P$, the window stays the same; the window rotates as $vs$ rotates about $v$). We build the arrangement of the curtains in $\rho$, and repeat it for all cells $\rho$ of $\mathcal{O}^2$. In the obtained decomposition (cells of $\mathcal{O}^2$, decomposed by the curtains) the link distance between any pair of points $(s, t)$ from one cell is the same: the link distance between $s$ and $t$ may change only if $t$ crosses a window of LDM($s$), which happens only if $t$ (the last two coordinates of point $(s, t)$ in 4d) crosses a curtain. To achieve $O(\log n)$ query time, preprocess the decomposition for point location (e.g., in the same way as was done for geodesic queries in [24, Theorem 3.1]).

### 5.4.8 Triple points

What remains is to account for the new cases (windows arrangement changes due to three windows passing through a common point). In Section 5.4.10 we give an algorithm to compute the locations $\mathcal{S} \subset P$ for $s$ such that three windows in LDM($s$) may intersect in a common point $t$ ($\mathcal{S}$ consists of curves of constant algebraic degree). The set $\mathcal{S}$ is overlaid with $\mathcal{O}$. By construction, for all points $s$ in one cell of the obtained 2d overlay $\mathcal{O}^*$, LDM($s$) is the same combinatorially: it has the same windows and they form the same arrangement (before the arrangement changes, three windows must pass through a common point, at which moment $s$ is in $\mathcal{S}$). Our final data structure, the (full) 2d LDM equivalence decomposition ("full" in the sense that it accounts for both simple and new cases) is built from $\mathcal{O}^*$ in the same way as the data structure for handling simple cases was built from $\mathcal{O}$ (Section 5.4.6).

As a by-product of computing $\mathcal{S}$, we also identify the corresponding locations for the "triple points" $t$, thus obtaining the (super)set $\mathbb{S} \subset P \times P$ of pairs $(s, t)$ of potential triple points. Then, (super)set $\mathbb{S}$ is overlaid with $\mathcal{O}^2$ subdivided by the curtains (Section 5.4.7). For all $(s, t)$ in one cell of the obtained 4d overlay $\hat{\mathcal{O}}$, the $s$-$t$ distance is the same. Preprocessing $\hat{\mathcal{O}}$ for point location, we obtain our other data structure: the (full) 4d link distance equivalence decomposition ("full" in the sense that it accounts for both simple and new cases).

### 5.4.9 Tracking bash-bash windows

To find the set $\mathcal{S}$ of sources potentially having triple points in their LDMs, we build yet another decomposition and do some preparation work. In [63], the combinatorial type of a window $w$ was defined as the pair $(v, e)$ where $v$ is the vertex of $P$ at which $w$ begins and $e$ is the edge on which the window ends. We refine the definition by saying that the window's combinatorial type is a pair $(v, a)$ where $a$ is the atomic segment on which $w$ ends. We decompose $P$ by rays from every atomic segment endpoint through every vertex visible to the endpoint (recall that vertices of $P$ are also endpoints of atomic segments). For any point $s$ in a cell of this decomposition $\mathcal{D}$ and every vertex $v$ visible to $s$, the atomic segment $a$ hit by the ray $sv$ is the same; let $s' = s'(s) \in a$ be the point where the ray hits the segment (we write $s' = s'(s)$ to emphasize its dependence on $s$). Since $a$ fully belongs to a single cell of the overlay $\mathcal{O}$, the map LDM($s'$) from any point $s' \in a$ has combinatorially the same bash-bash windows; moreover, the dependence of these windows on $s'$ (and hence on $s$) is known via the projection functions. We thus write $w = w(s)$ for a window $w$ in LDM($s'$).

### 5.4.10 Intersecting three windows

We are now ready to compute the (super)set $\mathcal{S}$ of points $s$ for which LDM($s$) may have a triple point $t$; the corresponding locations for $t$ (i.e., the set $\mathbb{S} \subset P \times P$ for the pairs $(s, t)$ defining the 4d link distance equivalence decomposition) are computed as well. We emphasize that we do not claim that LDM($s$) has a triple point for every $s$ in $\mathcal{S}$ (or that $\forall (s, t) \in \mathbb{S}$ LDM($s$) has a triple point at $t$); we only claim that $\mathcal{S}$ is sufficiently rich to "catch" all possible (sources of) maps with triple points (and that $\mathbb{S}$ includes all pairs of triple points).

Let $w_1, w_2, w_3$ be windows of LDM($s$) that intersect at $t$, of which at least 1 window rotates (so that LDM($s$) changes combinatorially when the three windows intersect at $t$). Recall from Observation 28 that the (super)set $W_{\text{EVG}}$ of possible static windows in LDM($s$) does not depend on $s$. We make 3 different guesses on how many of the three windows $w_1, w_2, w_3$ are rotating (1, 2, or all 3), and do different things for each of the guesses (Figure 5.8):

- Assuming only $w_1$ is rotating, LDM($s$) changes when $w_1$ passes through $t = w_2 \cap w_3$ (the intersection point of two static windows $w_2, w_3$). At this point there is a bash path between $s$ and $t$, implying that $s$ is on a bash-bash window of LDM($t$). We thus build LDM($t$) from every intersection point $t$ of two (potential) static windows $w_2, w_3 \in W_{\text{EVG}}$ and add to $\mathcal{S}$ all bash-bash windows of LDMs; the 4d set $\mathbb{S}$ is correspondingly appended with the Cartesian product of $t$ and all bash-bash windows of LDM($t$).
- Assuming 2 windows (say, $w_1, w_2$) are rotating, LDM($s$) changes when their intersection point $t$ passes through $w_3$. At this point there are two bash paths between $s$

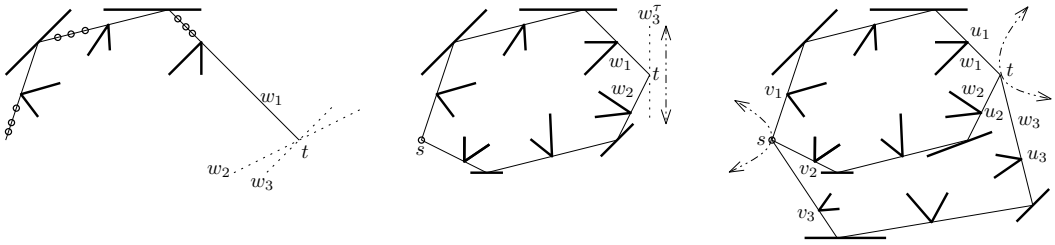



Figure 5.8: $P$ is bold, static windows from $W_{\mathsf{EVG}}$ are dotted, bash paths are solid. Left: LDM($s$) may change combinatorially when $s$ crosses a bash-bash window of LDM($t$) where $t = w_1 \cap w_2$ for some static windows $w_1, w_2 \in W_{\mathsf{EVG}}$ of LDM($s$); some possible locations for such $s$ are shown with hollow circles (if $s$ is moved slightly off the windows, $w_1$ stops going through $t$). Middle: LDM($s$) may change when there are two bash paths between $s$ and $t \in w_3$. Right: Each of $s$, $t$ moves on a curve so as to remain connected by 3 bash paths and be triple points in LDMs from each other.

and $t$ (in one path $t \in w_1$, in the other $t \in w_2$). That is, $s$ is the intersection point of two rotating windows in LDM($t$) (one window belonging to the bash path from $t$ that starts from following $w_1$, the other starts from following $w_2$). We thus take every window $w_3 \in W_{\mathsf{EVG}}$ as the potential static window in LDM($s$) and intersect it with every cell $\tau$ of $\mathcal{O}$; let $w_3^\tau$ denote part of $w_3$ inside $\tau$. For all points $t \in w_3^\tau$, LDM($t$) has the same set of windows, and in particular, the same set of bash-bash windows; each window is a known function of $t$ (via the projection functions). For every pair of the bash-bash windows (with the same link distance from $t$) in LDM($t$) we add to $\mathcal{S}$ the curve traced by their intersection as $t$ varies along $w_3^\tau$; the 4d set $\mathbb{S}$ is correspondingly appended with the Cartesian product of $w_3^\tau$ and the intersection points for $t \in w_3^\tau$.

- Assuming all three windows are rotating, we go through all pairs of cells $\sigma, \tau$ in $\mathcal{D}$, guessing that $s \in \sigma, t \in \tau$. In addition, we go through all triples $v_1, v_2, v_3$ of vertices visible from $s$ and all triples $u_1, u_2, u_3$ of vertices visible to $t$, guessing that the vertices support the first and the last windows respectively of the 3 bash paths (ending with the windows $w_1, w_2, w_3$) between $s$ and $t$. Since vertices of $P$ are endpoints of atomic segments, the decomposition $\mathcal{D}$ includes chords of the EVG, implying that the set of vertices visible to any point in a cell of $\mathcal{D}$ is the same: it is thus valid to speak about a triple of vertices visible to $s \in \sigma$ or to $t \in \tau$ without specifying the exact locations of $s$ and $t$ in the cells.

  Using the projection functions, we know whether for some number $k$ of links there indeed exist 3 length-$k$ bash paths from $s$, with the first links of the paths supported by $v_1, v_2, v_3$ and last links supported by $u_1, u_2, u_3$. If yes, let $w_1(s), w_2(s), w_3(s)$ be these last links (windows of LDM($s$)). For $t$ to be a triple point, the system

$$\begin{cases} t \in w_1(s) \\ t \in w_2(s) \\ t \in w_3(s) \end{cases} \tag{5.1}$$

  of 3 equations (each involving the projection function) with 4 unknowns (the coordinates of $s$ and $t$) must be satisfied (for $s \in \sigma, t \in \tau$). We solve the system and obtain the decompositions both for the 2d LDM equivalence decomposition (the set $\mathcal{S} \subseteq P$ such that LDM($s$) may change combinatorially only when $s$ is crossing $\mathcal{S}$, see Figure 5.8 for an example of such a combinatorial change) and for the 4d link

distance equivalence decomposition (the surface $\mathbb{S} \subseteq P^2$ such that the link distance between $s$ and $t$ may change only when $(s, t)$ is crossing $\mathbb{S}$). That is, the 4d set $\mathbb{S}$ consists of pairs $(s, t)$ satisfying (5.1), while for the 2d set $\mathcal{S}$ we take only the first two coordinates of the 4d pairs $(s, t)$ (i.e., the coordinates for $s$; the knowledge of corresponding locations for $t$ is ignored).

### 5.4.11 Putting things together

We summarize the steps described above, needed to build our data structures:

1. Build the VG and EVG (Section 5.4.1).
2. Designate all chords of the EVG as (static) windows in any LDM ever built (Section 5.4.2).
3. Compute the set $W_{\text{EVG}}$ of windows in LDMs from all chords of the EVG (any static window in any LDM is part of $W_{\text{EVG}}$) and compute the overlay $\mathcal{O}$ of the LDMs (Section 5.4.4).
4. Identify atomic segments by intersecting $\mathcal{O}$ with the boundary of $P$ (Section 5.4.5).
5. For each atomic segment $a$, compute the projection functions specifying the windows in LDM($s$) $\forall s \in a$ (Section 5.4.5). The functions are used to identify the 2d set $\mathcal{S} \subset P$ of potential triple points and the 4d set $\mathbb{S} \subset P^2$ of potential pairs of triple points of LDMs, as well as to answer link distance queries with parametric LDMs.
6. Build the decomposition $\mathcal{D}$ of $P$ by using rays from atomic segments though vertices of $P$ (Section 5.4.9). $\mathcal{D}$ is an auxiliary structure: we do not overlay $\mathcal{D}$ with any other decomposition and do not use it in our final data structures; $\mathcal{D}$ is used only to compute the 2d set $\mathcal{S} \subset P$ of potential triple points and the 4d set $\mathbb{S} \subset P^2$ of potential pairs of triple points – these sets will be used in our data structures, refining $\mathcal{O}$ and $\mathcal{O}^2$ respectively.
7. Compute the sets $\mathcal{S}$ and $\mathbb{S}$ for triple points: build LDMs from intersection points of windows from $W_{\text{EVG}}$, track intersections of windows in LDM($t$) as $t$ varies along each window from $W_{\text{EVG}}$, go through all triples of windows in pairs of cells of $\mathcal{D}$ (Section 5.4.10).
8. Build the overlay $\mathcal{O}^*$ of $\mathcal{O}$ with $\mathcal{S}$ (Section 5.4.8). To obtain the 2d LDM equivalence decomposition data structure, preprocess $\mathcal{O}^*$ for parametric point location, e.g., by further decomposing the cells of $\mathcal{O}^*$ so that for all $s$ in one cell of the refined decomposition, the ordering of $x$-coordinates of vertices of LDM($s$) is the same (as done in Section 5.4.6 for $\mathcal{O}$).
9. Decompose $\mathcal{O} \times \mathcal{O}$ by curtains (Section 5.4.7) and overlay the decomposition with $\mathbb{S}$, getting 4d decomposition $\hat{\mathcal{O}}$ (Section 5.4.8). To obtain the 4d distance link equivalence data structure, preprocess $\hat{\mathcal{O}}$ for point location queries.

**Theorem 29.** *2-point link distance queries in $P$ can be answered in $O(\log n)$ time after polynomial-time preprocessing. Link diameter and radius of $P$ can be found in poly time.*

# Chapter 6
# Discussion

This thesis studies algorithmic questions about movement through three recurring algorithmic lenses: *optimality*, *plausibility*, and *simplicity*. Concretely, the chapters address budgeted reward-maximizing routing (Orienteering/Touring), probabilistic interpolation of sparse trajectories (Random Walks), and turn-minimizing motion planning in polygonal domains (Minimum-Link Paths). This discussion summarizes the main results, reflects on practical considerations, and highlights open problems that can be considered for future research.

## 6.1 Budgeted routing: Orienteering and Touring

Chapter 2 studies the Orienteering Problem and several temporal extensions, with a focus on how underlying graph structure changes algorithmic complexity. A central takeaway is that, for the Orienteering Problem, restricting the graph to a path- or cycle-like structure is not by itself enough to guarantee tractability once time windows are introduced.

For the Orienteering Problem with time windows, we obtain a sharp contrast between undirected and directed graph classes. On undirected paths, even the single-window variant (OP-1TW) is already NP-hard (via the connection to line-TSP), and therefore the same holds for multiple windows. In contrast, however, on directed paths the structure of feasible walks can be exploited algorithmically: we give a dynamic program for OP-MTW running in $\mathcal{O}(m \log m)$ time, where $m$ is the total number of time windows. This highlights that the main source of difficulty is the ability to revisit vertices in different ways.

This difficulty of revisiting is additionally highlighted when we shift the focus to directed cycles. They allow revisits, but only in one direction, which creates enough structure for nontrivial algorithmic results whilst still permitting recurring visits of vertices that are absent on directed paths. For the covering version with a single time window per vertex (COP-1TW), we give a polynomial-time algorithm. Moreover, for the optimization version (OP-1TW), the exact complexity remains open, but we establish several positive results: a 2-approximation, an FPT algorithm parameterized by the number of long windows, and a PTAS. At the same time, allowing multiple time windows per vertex changes the complexity fundamentally: COP-MTW (and thus OP-MTW) is NP-hard even on directed cycles with only unit profit and edge cost. Taken together, these results show that the distinction between one and multiple time windows is algorithmically decisive already on very restricted graph classes.

We also consider a different temporal generalization in which *edge availability* varies over time (dynamic graphs). Although this model is closely related in spirit to vertex time windows, the complexity landscape is completely different. In particular, the Orienteering Problem on dynamic undirected paths admits a simple polynomial-time algorithm, whereas the problem is NP-hard on dynamic trees even with unit profits and unit costs. This contrast further supports the broader theme of the chapter: small modeling changes in temporal feasibility can lead to qualitatively different algorithmic behavior.

Finally, for the classical Orienteering Problem without time windows, we show that exact polynomial-time solvability cannot be expected even on trees with unit profit and unit edge cost (and hence also not on bounded-treewidth graphs), but bounded treewidth still enables strong positive approximation results. We present a dynamic program for unit profits on bounded-treewidth graphs and extend it to a $(1+\varepsilon)$-approximation for arbitrary profits. This improves substantially over the best known approximation guarantee for general graphs and demonstrates that treewidth is a useful structural parameter for obtaining near-optimal solutions in budgeted routing.

In Chapter 3, we shift the focus from vertex rewards to edge rewards and from route length to route quality measures such as area coverage. First, we give an improved approximation algorithm for the Edge Orienteering Problem, which is a special case of the Touring Problem. Then, we present a prototyping framework for customizable tour planning, in which parameter values can be adjusted to meet users' individual preferences and multiple planning algorithms can be integrated and compared. This framework is a first step towards more flexible and user-centric tour planning tools, and it opens up many possibilities for future work on both the algorithmic and user-interface sides.

**Future Work.** Several natural questions remain open. The most prominent one is the exact complexity of OP-1TW on directed cycles. A natural direction is to better understand the role of *time-window structures* as a parameter. Our FPT result for directed cycles parametrized by the number of long windows suggest that different parameters (e.g. number of rounds or overlap complexity of windows) are worth looking into. Such parameters may lead to improved exact or approximation algorithms that will give us better understanding of the intricacies of the problem and lead to either a hardness result or a parameter-free exact algorithm.

For the classical Orienteering Problem without time windows, an important open question is how far the bounded-treewidth approximation approach can be pushed. In particular, it is natural to ask whether the $(1+\varepsilon)$-approximation can be extended from bounded-treewidth graphs to larger sparse graph classes, such as planar or minor-free graphs, possibly via decomposition techniques. Since related prize-collecting routing problems admit PTAS-type results on planar graphs, this appears to be a promising direction.

More broadly, this chapter shows that temporal constraints can make the problem even on simple graph classes difficult, but also that carefully chosen structural restrictions (directionality, bounded treewidth, limited revisitation patterns) can enable strong algorithmic results. A systematic understanding of which combinations of graph structure and temporal constraints lead to tractability remains an appealing goal for future work.

In the context of the Touring Problem, there are many directions for future work on both the algorithmic and user-interface sides. On the algorithmic side, improving the approximation factor for Edge Orienteering and Touring is a natural question. It might prove hard to improve the approximation factor by improving the reduction from the Orienteering Problem, but a more direct approach to Edge Orienteering might yield better results. Of course, any improvement in the approximation factor for the Orienteering Problem would

immediately improve our approximation factor for Edge Orienteering and Touring, while also being interesting in its own right.

For the Touring Problem, there are many directions for future work with respect to the prototyping framework. For instance, it would be interesting to extend the framework to support more complex quality measures, such as user-defined functions of route features or multi-objective optimization. On the user-interface side, conducting user studies to evaluate the usability and effectiveness of the framework in real-world tour planning scenarios would be of interest. Additionally, integrating real-time data (e.g., traffic conditions, weather) and allowing for dynamic re-planning could further enhance the practical utility of the tool. Finally, there is a lot of ground for exploring more algorithms for the Touring Problem, both exact and approximate, and for comparing their performance in different settings. This could include algorithms that leverage specific graph structures and heuristics for large-scale instances.

## 6.2 Discretized Random Walks

Chapter 4 addresses the second theme of this thesis, *plausibility*, by studying how to interpolate sparse trajectories with movement models that remain faithful to stochastic movement assumptions while still being computationally tractable. The central contribution is an algorithmic framework for *conditioned* random-walk interpolation on discretized space-time: given two time-stamped locations and a one-step transition model, the framework computes quantities about all feasible interpolating walks and can also sample individual walks that exactly satisfy the endpoint constraints.

A key conceptual contribution of the chapter is the separation between *model specification* and *conditioning/inference*. Movement behavior is encoded locally through transition matrices (or families of transition matrices), while conditioning on start and end measurements is handled by a dynamic program. Because of this separation it is possible to use the dynamic program across a broad range of models, from Brownian motion approximations to biased, correlated, and mixed state-dependent random walks. In particular, the framework extends beyond methods such as Brownian bridges that primarily yield marginal distributions, by additionally enabling sampling of individual conditioned walks and computing visit probabilities for target regions.

From an algorithmic point of view, the chapter shows that discretization provides a useful middle ground between very simple grid-walk models and continuous-state simulation methods. The forward and backward dynamic programs yield transition densities and conditioned utilization distributions efficiently, and the backtracking-based WALKER procedure guarantees that sampled trajectories are consistent with the observed endpoints whenever a feasible path exists. This exact conditioning is an important distinction from heuristic attraction-based approaches (such as RTG-style methods), which may produce qualitatively plausible trajectories but do not in general sample from the intended conditioned movement model.

Another important contribution is the extension from endpoint-conditioned interpolation to calculate *visit probabilities* and *utilization distributions*. These quantities are often the actual object of interest in downstream analyses (e.g., exposure risk, habitat use, home range estimation), and the chapter demonstrates that they can be computed directly from the same dynamic-programming tables rather than estimated only through repeated simulation. This substantially improves interpretability and makes the framework useful not only as a trajectory generator but also as an inference tool for spatial analysis.

The chapter also demonstrates that the framework naturally supports *heterogeneous* movement models. By allowing transition kernels to depend on location, time, and (discretized) heading, it can represent mixed and correlated random walks as well as simple state-space style models. This is practically significant: many realistic movement processes are neither purely diffusive nor stationary, and habitat- or time-dependent transitions are often essential to capture behavior.

At the same time, this flexibility comes with computational and modeling trade-offs. The state augmentation used for correlated walks increases the dimensionality of the dynamic program, and the runtime grows quickly with grid size, time horizon, step radius, and number of states/direction bins. This is visible in the home-range experiments, where the proposed method is much slower than simplified BRB, even though it provides a more direct incorporation of habitat into the movement model. Thus, as in many probabilistic inference settings, improved model fidelity and richer outputs come at the cost of higher computational effort.

The ecological case study on Eastern Bettongs illustrates both the promise and the limitations of the approach in applied settings. On the positive side, the framework supports state-dependent habitat-use analysis from sparse trajectories by translating step-wise uncertainty into utilization distributions, rather than treating movement between GPS fixes as straight-line segments. This enables a more nuanced notion of habitat use during movement and provides an interpretable baseline-vs-use comparison. The observed contrasts (especially the clearer male foraging–fast contrast) suggests that canopy cover contains a detectable signal, but the wider confidence intervals in other contrasts also highlight that inference remains sensitive to data availability, inter-individual variability, and model choices.

**Future Work.** Several directions appear particularly promising. First, on the algorithmic side, improving scalability is an immediate goal. Possible avenues include FFT-based convolution for large kernels, GPU implementations, adaptive or multiresolution grids, and pruning schemes that restrict computation to low-probability regions. Such improvements would be especially valuable for long trajectories, large rasters, or richer state spaces.

Second, the framework could be extended toward richer latent-state models. In the current setup, state-dependent transitions can be prescribed from external information (e.g., habitat or known behavioral states), but in many applications the behavioral state is itself uncertain. Integrating hidden-state inference (e.g., HMM-style state uncertainty) with endpoint-conditioned interpolation is a natural and challenging next step.

Third, parameter estimation and uncertainty propagation deserve further attention. The chapter demonstrates how to build transition matrices from observed displacements (including via Gaussian mixtures), but the downstream effect of parameter uncertainty on visit probabilities, utilization distributions, and home-range estimates is not yet quantified. Bootstrap or Bayesian approaches that propagate this uncertainty through the dynamic programs could make the resulting ecological conclusions more robust.

Finally, there is substantial potential for broader applications. The same computational primitives used here for habitat-use analysis could be adapted to disease exposure estimation, route compliance analysis, probabilistic corridor detection, and path reconstruction under spatial constraints. A systematic comparison with trajectory-recovery methods learned from large datasets would also be valuable, especially in regimes with sparse data where model-based interpolation may retain an advantage.

## 6.3 Minimum-Link Paths

Chapter 5 studies minimum-link paths in polygonal domains through the "simplicity" lens, where the cost of motion is the number of turns rather than Euclidean length. The setting is deceptively subtle: unlike geodesic paths, minimum-link paths do not admit a clean discretization, bend points can have high bit complexity even in simple polygons, and in domains with holes it is not obvious how to control the combinatorial complexity of reachable regions and distance maps.

The main contribution of the chapter is to show that, despite these complications, the core two-point and global problems under link distance are polynomial-time solvable. We construct data structures for two-point link distance queries with $O(\log n)$ query time after polynomial-time preprocessing, by building link-distance analogues of the 2d and 4d equivalence decompositions known for geodesic queries. The technical backbone is a parametric view of link distance maps (LDMs): we track which windows are static versus rotating, and we refine the decomposition further to account for the events that do not arise in simple polygons, in particular changes in the window arrangement caused by triple-point intersections. Once these two-point structures are available, the link diameter and link radius/center follow via MetaTheorem 26, filling the remaining gap in the diameter/radius landscape for polygonal domains with holes.

**Future Work.** A natural next step is to improve the efficiency of the algorithms developed in Chapter 5. The main goal of the chapter is to establish polynomial-time solvability, not to optimize the resulting bounds, and there is substantial room for improvement in both preprocessing time and structural complexity. In particular, it is valuable to simplify the decompositions underlying the link-distance query structures and to obtain a more concise description of the geometric events specific to domains with holes.

A more conceptual direction is to seek a better structural understanding of link distance itself. One major issue is that minimum-link paths resist clean discretization, and much of these technical difficulties stem from the interaction of rotating windows and higher-order events such as triple intersections. It would therefore be interesting to determine whether these phenomena can be organized into a smaller collection of canonical events, or whether an alternative representation of link distance maps could avoid some of this complexity altogether.

# Bibliography

## Own Publications


[MH1] A. Atak, K. Buchin, M. Hagedoorn, J. Heinrichs, K. Hogreve, G. Li, and P. Pawelczyk. "Computing Maximum Polygonal Packings in Convex Polygons Using Best-Fit, Genetic Algorithms and ILPs (CG Challenge)". en. In: vol. 293. 2024, 83:1–83:9. DOI: 10.4230/LIPICS.SOCG.2024.83.

[MH2] K. Buchin, J. Conradi, L. Deryckere, M. Hagedoorn, and C. Rehs. "Tight Lower Bound for Approximating (k, ℓ)-Center Clustering under the Fréchet Distance". In: *21st Spanish Meeting on Computational Geometry (EGC25)*. 2025.

[MH3] K. Buchin and M. Hagedoorn. "Improved approximation algorithm for the edge orienteering problem". In: *Information Processing Letters* 195 (July 2026), p. 106660. ISSN: 0020-0190. DOI: 10.1016/j.ipl.2026.106660.

[MH4] K. Buchin, M. Hagedoorn, and A. Korn. "Discretized Random Walk Models for Efficient Movement Interpolation". In: *Proceedings of the 32nd ACM International Conference on Advances in Geographic Information Systems*. SIGSPATIAL '24. Oct. 2024, pp. 102–112. DOI: 10.1145/3678717.3691231.

[MH5] K. Buchin, M. Hagedoorn, and G. Li. "Tour4Me: a framework for customized tour planning algorithms". In: *Proceedings of the 30th International Conference on Advances in Geographic Information Systems*. SIGSPATIAL '22. Nov. 2022, pp. 1–4. DOI: 10.1145/3557915.3560992.

[MH6] K. Buchin, M. Hagedoorn, G. Li, and C. Rehs. "Orienteering (with Time Windows) on Restricted Graph Classes". In: *SOFSEM 2025: Theory and Practice of Computer Science*. 2025, pp. 151–165. DOI: 10.1007/978-3-031-82670-2_12.

[MH7] M. Hagedoorn and V. Polishchuk. "Link Diameter, Radius and 2-Point Link Distance Queries in Polygonal Domains". en. In: vol. 349. 2025, 34:1–34:15. DOI: 10.4230/LIPICS.WADS.2025.34.


## References


[1] H. G. Abeledo, R. Fukasawa, A. A. Pessoa, and E. Uchoa. "The time dependent traveling salesman problem: polyhedra and algorithm". In: *Mathematical Programming Computation* 5.1 (Sept. 2013), pp. 27–55. ISSN: 1867-2957. DOI: 10.1007/S12532-012-0047-Y.

[2] A. Aggarwal, H. Booth, J. O'Rourke, S. Suri, and C. K. Yap. "Finding minimal convex nested polygons". In: *Information and Computation* 83.1 (Oct. 1989), pp. 98–110. ISSN: 0890-5401. DOI: 10.1016/0890-5401(89)90049-7.

[3] H.-K. Ahn, L. Barba, P. Bose, J.-L. De Carufel, M. Korman, and E. Oh. “A Linear-Time Algorithm for the Geodesic Center of a Simple Polygon”. In: *Discrete & Computational Geometry* 56.4 (July 2016), pp. 836–859. ISSN: 1432-0444. DOI: 10.1007/s00454-016-9796-0.

[4] C. Archetti, M. G. Speranza, and D. Vigo. “Vehicle routing problems with profits”. In: *Vehicle routing: Problems, methods, and applications, second edition*. Nov. 2014, pp. 273–297. DOI: 10.1137/1.9781611973594.ch10.

[5] E. M. Arkin, J. S. Mitchell, and S. Suri. “Logarithmic-time link path queries in a simple polygon”. In: *International Journal of Computational Geometry & Applications* 5.04 (Dec. 1995), pp. 369–395. ISSN: 1793-6357. DOI: 10.1142/S0218195995000234.

[6] E. Arseneva, M.-K. Chiu, M. Korman, A. Markovic, Y. Okamoto, A. Ooms, A. Van Renssen, and M. Roeloffzen. “Rectilinear link diameter and radius in a rectilinear polygonal domain”. In: *Computational Geometry* 92 (Jan. 2021), p. 101685. ISSN: 0925-7721. DOI: 10.1016/j.comgeo.2020.101685.

[7] S. W. Bae, M. Korman, and Y. Okamoto. “Computing the geodesic centers of a polygonal domain”. In: *Computational Geometry* 77 (Mar. 2019), pp. 3–9. ISSN: 0925-7721. DOI: 10.1016/j.comgeo.2015.10.009.

[8] S. W. Bae, M. Korman, and Y. Okamoto. “The Geodesic Diameter of Polygonal Domains”. In: *Discrete & Computational Geometry* 50.2 (July 2013), pp. 306–329. ISSN: 1432-0444. DOI: 10.1007/s00454-013-9527-8.

[9] N. Bansal, A. Blum, S. Chawla, and A. Meyerson. “Approximation algorithms for deadline-TSP and vehicle routing with time-windows”. In: *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*. STOC04. June 2004, pp. 166–174. DOI: 10.1145/1007352.1007385.

[10] S. Benhamou. “Dynamic Approach to Space and Habitat Use Based on Biased Random Bridges”. en. In: *PLOS ONE* 6.1 (Jan. 2011). Publisher: Public Library of Science, e14592. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0014592.

[11] S. de Berg, T. Miltzow, and F. Staals. “Towards Space Efficient Two-Point Shortest Path Queries in a Polygonal Domain”. en. In: *40th International Symposium on Computational Geometry (SoCG 2024)*. Vol. 293. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. 2024, 17:1–17:16. DOI: 10.4230/LIPIcs.SoCG.2024.17.

[12] L. Bigras, M. Gamache, and G. Savard. “The time-dependent traveling salesman problem and single machine scheduling problems with sequence dependent setup times”. In: *Discrete Optimization* 5.4 (Nov. 2008), pp. 685–699. ISSN: 1572-5286. DOI: 10.1016/J.DISOPT.2008.04.001.

[13] A. Blum, S. Chawla, D. R. Karger, T. Lane, A. Meyerson, and M. Minkoff. “Approximation Algorithms for Orienteering and Discounted-Reward TSP”. In: *SIAM Journal on Computing* 37.2 (Jan. 2007), pp. 653–670. ISSN: 1095-7111. DOI: 10.1137/050645464.

[14] H. L. Bodlaender. “A Linear-Time Algorithm for Finding Tree-Decompositions of Small Treewidth”. In: *SIAM Journal on Computing* 25.6 (Dec. 1996), pp. 1305–1317. ISSN: 1095-7111. DOI: 10.1137/S0097539793251219.

[15] G. Boeing. “OSMnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks”. In: *Computers, Environment and Urban Systems* 65 (Sept. 2017), pp. 126–139. ISSN: 0198-9715. DOI: 10.1016/j.compenvurbsys.2017.05.004.

[16] L. Börger, N. Franconi, G. De Michele, A. Gantz, F. Meschi, A. Manica, S. Llovari, and T. Coulson. “Effects of sampling regime on the mean and variance of home range size estimates”. In: *Journal of Animal Ecology* 75.6 (Sept. 2006), pp. 1393–1405. ISSN: 1365-2656. DOI: 10.1111/j.1365-2656.2006.01164.x.

[17] K. Buchin and A. Popov. *Efficiently Computing Bridgelets*.

[18] K. Buchin, S. Sijben, E. van Loon, N. Sapir, S. Mercier, T. J. Marie Arseneau, and E. Willems. “Deriving movement properties and the effect of the environment from the Brownian bridge movement model in monkeys and birds”. In: *Movement Ecology* 3.1 (June 2015), p. 18. ISSN: 2051-3933. DOI: 10.1186/s40462-015-0043-8.

[19] W. H. Burt. “Territoriality and Home Range Concepts as Applied to Mammals”. In: *Journal of Mammalogy* 24.3 (Aug. 1943), pp. 346–352. ISSN: 0022-2372. DOI: 10.2307/1374834.

[20] C. Calenge, contributions from Stephane Dray, and M. Royer. *adehabitatLT: Analysis of Animal Movements*. R package version 0.3.27. 2023.

[21] C. Chekuri, N. Korula, and M. Pál. “Improved algorithms for orienteering and related problems”. In: *ACM Transactions on Algorithms* 8.3 (July 2012), pp. 1–27. ISSN: 1549-6333. DOI: 10.1145/2229163.2229167.

[22] K. Chen and S. Har-Peled. “The Euclidean Orienteering Problem Revisited”. In: *SIAM Journal on Computing* 38.1 (Jan. 2008), pp. 385–397. ISSN: 1095-7111. DOI: 10.1137/060667839.

[23] Y. Chen, G. Cong, and C. Anda. “TERI: An Effective Framework for Trajectory Recovery with Irregular Time Intervals”. In: *Proceedings of the VLDB Endowment* 17.3 (Nov. 2023), pp. 414–426. ISSN: 2150-8097. DOI: 10.14778/3632093.3632105.

[24] Y.-J. Chiang and J. S. Mitchell. “Two-point Euclidean shortest path queries in the plane”. In: *Proceedings of the 10th ACM-SIAM Symposium on Discrete Algorithms*. Citeseer. 1999, pp. 215–224. DOI: 10.5555/314500.314560.

[25] E. Codling, M. Plank, and S. Benhamou. “Random walk models in biology”. In: *Journal of The Royal Society Interface* 5.25 (Apr. 2008). Publisher: Royal Society, pp. 813–834. ISSN: 1742-5662. DOI: 10.1098/rsif.2008.0014.

[26] M. Cygan, F. V. Fomin, Ł. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Vol. 5. Springer, 2015. DOI: 10.1007/978-3-319-21275-3.

[27] Department of the Environment, Tourism, Science and Innovation, Queensland Government. *Seasonal Persistent Green - Landsat, JRSRP Algorithm Version 3.0, Australia Coverage*. Version 3.0. The Creative Commons Attribution 4.0 International (CC BY 4.0) license. 2022.

[28] H. N. Djidjev, A. Lingas, and J.-R. Sack. “An $O(n \log n)$ algorithm for computing the link center of a simple polygon”. In: *Discrete & Computational Geometry* 8.2 (July 1992), pp. 131–152. ISSN: 1432-0444. DOI: 10.1007/BF02293040.

[29] H. Edelsbrunner, L. J. Guibas, and J. Stolfi. “Optimal point location in a monotone subdivision”. In: *SIAM Journal on Computing* 15.2 (1986), pp. 317–340. DOI: 10.1137/0215023.

[30] D. Elias and B. Kuijpers. “Visit Probability in Space-Time Prisms Based on Binomial Random Walk”. In: *ISPRS International Journal of Geo-Information* 9.9 (Sept. 2020), p. 555. ISSN: 2220-9964. DOI: 10.3390/IJGI9090555.

[31] D. Feillet, P. Dejax, and M. Gendreau. “Traveling Salesman Problems with Profits”. In: *Transportation Science* 39.2 (May 2005), pp. 188–205. ISSN: 1526-5447. DOI: 10.1287/trsc.1030.0079.

[32] M. Fischetti, J. J. S. González, and P. Toth. “Solving the Orienteering Problem through Branch-and-Cut”. In: *INFORMS Journal on Computing* 10.2 (May 1998), pp. 133–148. ISSN: 1526-5528. DOI: 10.1287/ijoc.10.2.133.

[33] B. Fristedt and L. Gray. *A Modern Approach to Probability Theory*. eng. Probability and its applications. Boston: Birkhäuser, 1997. DOI: 10.1007/978-1-4899-2837-5.

[34] R. Gardiner, R. Hamer, V. Leos-Barajas, C. Peñaherrera-Palma, M. E. Jones, and C. Johnson. “State-space modeling reveals habitat perception of a small terrestrial mammal in a fragmented landscape”. In: *Ecology and Evolution* 9.17 (Aug. 2019), pp. 9804–9814. ISSN: 2045-7758. DOI: 10.1002/ece3.5519.

[35] N. Garg, S. Khanna, and A. Kumar. “Hardness of Approximation for Orienteering with Multiple Time Windows”. In: *SODA ’21*. Jan. 2021, pp. 2977–2990. DOI: 10.1137/1.9781611976465.177.

[36] D. Gavalas, C. Konstantopoulos, K. Mastakas, G. Pantziou, and N. Vathis. “Approximation algorithms for the arc orienteering problem”. In: *Information Processing Letters* 115.2 (Feb. 2015), pp. 313–315. ISSN: 0020-0190. DOI: 10.1016/j.ipl.2014.10.003.

[37] R. Geisberger, P. Sanders, D. Schultes, and D. Delling. “Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks”. In: *Experimental Algorithms*. 2008, pp. 319–333. DOI: 10.1007/978-3-540-68552-4_24.

[38] A. Gemsa, T. Pajor, D. Wagner, and T. Zündorf. “Efficient Computation of Jogging Routes”. In: *Experimental Algorithms*. Vol. 7933. June 2013, pp. 272–283. DOI: 10.1007/978-3-642-38527-8_25.

[39] H.-O. Georgii. *Stochastics: Introduction to Probability and Statistics*. De Gruyter, Nov. 2012. DOI: 10.1515/9783110293609.

[40] S. K. Ghosh. “Computing the visibility polygon from a convex set and related problems”. In: *Journal of Algorithms* 12.1 (Mar. 1991), pp. 75–95. ISSN: 0196-6774. DOI: 10.1016/0196-6774(91)90024-S.

[41] B. L. Golden, L. Levy, and R. Vohra. “The orienteering problem”. In: *Naval Research Logistics* 34.3 (June 1987), pp. 307–318. ISSN: 1520-6750. DOI: 10.1002/1520-6750(198706)34:3<307::aid-nav3220340302>3.0.co;2-d.

[42] L. J. Guibas and J. Hershberger. “Optimal shortest path queries in a simple polygon”. In: *Proceedings of the third annual symposium on Computational geometry*. SCG ’87. 1987, pp. 50–63. DOI: 10.1145/41958.41964.

[43] A. Gunawan, H. C. Lau, and P. Vansteenwegen. “Orienteering Problem: A survey of recent variants, solution approaches and applications”. In: *European Journal of Operational Research* 255.2 (Dec. 2016), pp. 315–332. ISSN: 0377-2217. DOI: 10.1016/J.EJOR.2016.04.059.

[44] J. Hershberger and J. Snoeyink. “Computing minimum length paths of a given homotopy class”. In: *Computational Geometry* 4.2 (June 1994), pp. 63–97. ISSN: 0925-7721. DOI: 10.1016/0925-7721(94)90010-8.

[45] J. Hershberger and S. Suri. “Matrix Searching with the Shortest-Path Metric”. In: *SIAM Journal on Computing* 26.6 (Dec. 1997), pp. 1612–1634. ISSN: 1095-7111. DOI: 10.1137/S0097539793253577.

[46] J. Horne, E. Garton, S. Krone, and J. Lewis. “Analyzing Animal Movements Using Brownian Bridges”. en. In: *Ecology* 88.9 (Sept. 2007), pp. 2354–2363. ISSN: 1939-9170. DOI: 10.1890/06-0957.1.

[47] G. James, D. Witten, T. Hastie, R. Tibshirani, and J. E. Taylor. *An Introduction to Statistical Learning: with Applications in Python*. eng. Springer texts in statistics. Cham, Switzerland: Springer, 2023. DOI: 10.1007/978-3-031-38747-0.

[48] S. R. Jammalamadaka and A. SenGupta. *Topics in Circular Statistics*. en. Vol. 5. Series on Multivariate Analysis. WORLD SCIENTIFIC, Apr. 2001. DOI: 10.1142/4031.

[49] S. Kahan and J. Snoeyink. "On the bit complexity of minimum link paths: Superquadratic algorithms for problem solvable in linear time". In: *Computational Geometry* 12.1–2 (Feb. 1999), pp. 33–44. ISSN: 0925-7721. DOI: 10.1016/S0925-7721(98)00041-8.

[50] M. Kamberg. *Algorithms for Round Trip Planning using Waypoints*. Master's thesis, not publicly available. Archived at Lehrstuhl 11, Fakültat für Informatik, Technische Universität Dortmund (copy on request). Dec. 2025.

[51] P. Kareiva and N. Shigesada. "Analyzing insect movement as a correlated random walk". en. In: *Oecologia* 56.2–3 (Feb. 1983), pp. 234–238. ISSN: 1432-1939. DOI: 10.1007/BF00379695.

[52] Y. Ke. "An efficient algorithm for link-distance problems". In: *Proceedings of the fifth annual symposium on Computational geometry*. SCG '89. 1989, pp. 69–78. DOI: 10.1145/73833.73841.

[53] N. Korula. *Approximation algorithms for network design and orienteering*. University of Illinois at Urbana-Champaign, 2010.

[54] I. Kostitsyna, M. Löffler, V. Polishchuk, and F. Staals. "On the complexity of minimum-link path problems". en. In: *Journal of Computational Geometry* 8.2 (2017). Special issue on SoCG'16, pp. 80–108. DOI: 10.20382/jocg.v8i2a5.

[55] B. Kranstauber, K. Safi, and F. Bartumeus. "Bivariate Gaussian bridges: directional factorization of diffusion in Brownian bridge models". In: *Movement Ecology* 2.1 (Mar. 2014), pp. 1–10. ISSN: 2051-3933. DOI: 10.1186/2051-3933-2-5.

[56] J. Krumm. "Maximum entropy bridgelets for trajectory completion". In: *Proceedings of the 30th International Conference on Advances in Geographic Information Systems*. SIGSPATIAL '22. New York, NY, USA, Nov. 2022, pp. 1–8. DOI: 10.1145/3557915.3561015.

[57] G. Laporte and S. Martello. "The selective travelling salesman problem". In: *Discrete Applied Mathematics* 26.2–3 (Mar. 1990), pp. 193–207. ISSN: 0166-218X. DOI: 10.1016/0166-218X(90)90100-Q.

[58] J. A. Long, T. A. Nelson, and F. S. Nathoo. "Toward a kinetic-based probabilistic time geography". In: *International Journal of Geographical Information Science* 28.5 (Sept. 2013), pp. 855–874. ISSN: 1362-3087. DOI: 10.1080/13658816.2013.818151.

[59] Y. Lu and C. Shahabi. "An arc orienteering algorithm to find the most scenic path on a large-scale road network". In: *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*. SIGSPATIAL'15. Nov. 2015, pp. 1–10. DOI: 10.1145/2820783.2820835.

[60] U. Lund and C. Agostinelli. *CircStats: Circular Statistics, from "Topics in Circular Statistics" (2001)*. R package version 0.2-6. 2018.

[61] J. Mitchell, C. Piatko, and E. M. Arkin. "Computing a shortest k-link path in a polygon". In: *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science*. IEEE. 1992, pp. 573–582. DOI: 10.1109/sfcs.1992.267794.

[62] J. Mitchell, V. Polishchuk, and M. Sysikaski. “Minimum-link paths revisited”. In: *Computational Geometry* 47.6 (Aug. 2014). Special issue on EuroCG’11, pp. 651–667. ISSN: 0925-7721. DOI: 10.1016/j.comgeo.2013.12.005.

[63] J. Mitchell, G. Rote, and G. J Woeginger. “Minimum-link paths among obstacles in the plane”. In: *Algorithmica* 8.1–6 (Dec. 1992), pp. 431–459. ISSN: 1432-0541. DOI: 10.1007/bf01758855.

[64] R. Nathan, W. M. Getz, E. Revilla, M. Holyoak, R. Kadmon, D. Saltz, and P. E. Smouse. “A movement ecology paradigm for unifying organismal movement research”. In: *Proceedings of the National Academy of Sciences* 105.49 (Dec. 2008), pp. 19052–19059. ISSN: 1091-6490. DOI: 10.1073/pnas.0800375105.

[65] R. Pace. “Chapter 7 - Estimating and Visualizing Movement Paths from Radio-Tracking Data”. In: *Radio Tracking and Animal Populations*. San Diego, Jan. 2001, pp. 189–206. DOI: 10.1016/B978-012497781-5/50008-6.

[66] C. S. Patlak. “Random walk with persistence and external bias”. en. In: *The Bulletin of Mathematical Biophysics* 15.3 (Sept. 1953), pp. 311–338. ISSN: 1522-9602. DOI: 10.1007/bf02476407.

[67] T. Patterson, L. Thomas, C. Wilcox, O. Ovaskainen, and J. Matthiopoulos. “State–-space models of individual animal movement”. In: *Trends in Ecology & Evolution* 23.2 (Feb. 2008), pp. 87–94. ISSN: 0169-5347. DOI: 10.1016/j.tree.2007.10.009.

[68] K. Pearson. “The Problem of the Random Walk”. en. In: *Nature* 72.1865 (July 1905), pp. 294–294. ISSN: 1476-4687. DOI: 10.1038/072294b0.

[69] R. Ramesh, Y.-S. Yoon, and M. H. Karwan. “An Optimal Algorithm for the Orienteering Tour Problem”. In: *ORSA Journal on Computing* 4.2 (May 1992), pp. 155–165. ISSN: 2326-3245. DOI: 10.1287/ijoc.4.2.155.

[70] K. Ren and M. R. Salavatipour. “Approximation Schemes for Orienteering and Deadline TSP in Doubling Metrics”. In: *arXiv preprint arXiv:2405.00818* (2024).

[71] J.-R. Sack and J. Urrutia. *Handbook of computational geometry*. Elsevier, 1999. DOI: 10.1016/B978-0-444-82537-7.X5000-1.

[72] *Sentinel-2 10-Meter Land Use/Land Cover*. Tech. rep.

[73] W. Souffriau, P. Vansteenwegen, J. Vertommen, G. V. Berghe, and D. V. Oudheusden. “A Personalized Tourist Trip Design Algorithm For Mobile Tourist Guides”. In: *Applied Artificial Intelligence* 22.10 (Oct. 2008), pp. 964–985. ISSN: 1087-6545. DOI: 10.1080/08839510802379626.

[74] F. Stavropoulou, P. P. Repoussis, and C. D. Tarantilis. “The Vehicle Routing Problem with Profits and consistency constraints”. In: *European Journal of Operational Research* 274.1 (Apr. 2019), pp. 340–356. ISSN: 0377-2217. DOI: 10.1016/j.ejor.2018.09.046.

[75] H. P. Steyn and L. Liebenberg. “The Art of Tracking. The Origin of Science”. In: *The South African Archaeological Bulletin* 45.152 (Dec. 1990), p. 137. ISSN: 0038-1969. DOI: 10.2307/3887981.

[76] P. Stroobant, P. Audenaert, D. Colle, and M. Pickavet. “Generating constrained length personalized bicycle tours”. In: *4OR* 16.4 (Jan. 2018), pp. 411–439. ISSN: 1614-2411. DOI: 10.1007/s10288-018-0371-9.

[77] H. Sun, C. Yang, L. Deng, F. Zhou, F. Huang, and K. Zheng. “PeriodicMove: Shift-aware Human Mobility Recovery with Graph Neural Network”. In: *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. CIKM ’21. New York, NY, USA, Oct. 2021, pp. 1734–1743. DOI: 10.1145/3459637.3482284.

[78] S. Suri. “A linear time algorithm for minimum link paths inside a simple polygon”. In: *Computer Vision, Graphics, and Image Processing* 35.1 (Apr. 1986), pp. 99–110. ISSN: 0734-189X. DOI: 10.1016/0734-189x(86)90070-8.

[79] S. Suri. “Computing geodesic furthest neighbors in simple polygons”. In: *Journal of Computer and System Sciences* 39.2 (Oct. 1989), pp. 220–235. ISSN: 0022-0000. DOI: 10.1016/0022-0000(89)90045-7.

[80] S. Suri. “On some link distance problems in a simple polygon”. In: *IEEE transactions on Robotics and Automation* 6.1 (1990), pp. 108–113. DOI: 10.1109/70.88124.

[81] S. Suri and J. O’Rourke. “Worst-case optimal algorithms for constructing visibility polygons with holes”. In: *Proceedings of the second annual symposium on Computational geometry*. SCG ’86. 1986, pp. 14–23. DOI: 10.1145/10515.10517.

[82] X. Tan. “Sweeping simple polygons with the minimum number of chain guards”. In: *Information Processing Letters* 102.2–3 (Apr. 2007), pp. 66–71. ISSN: 0020-0190. DOI: 10.1016/j.ipl.2006.11.010.

[83] G. Technitis, W. Othman, K. Safi, and R. Weibel. “From A to B, randomly: a point-to-point random trajectory generator for animal movement”. In: *International Journal of Geographical Information Science* 29.6 (June 2015). Publisher: Taylor & Francis, pp. 912–934. ISSN: 1365-8816. DOI: 10.1080/13658816.2014.999682.

[84] G. Technitis, R. Weibel, B. Kranstauber, and K. Safi. “An Algorithm for Empirically Informed Random Trajectory Generation Between Two Endpoints”. en. In: *International Conference on GIScience Short Paper Proceedings* 1.1 (2016). ISSN: 2573-783X. DOI: 10.21433/B31194g0c634.

[85] C. D. Toth, J. O’Rourke, and J. E. Goodman. *Handbook of discrete and computational geometry*. CRC press, 2017. DOI: 10.1201/9781315119601.

[86] Y. Tremblay, P. W. Robinson, and D. P. Costa. “A Parsimonious Approach to Modeling Animal Movement Data”. In: *PLoS ONE* 4.3 (Mar. 2009), e4711. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0004711.

[87] T. Tsiligirides. “Heuristic Methods Applied to Orienteering”. In: *Journal of the Operational Research Society* 35.9 (Sept. 1984), pp. 797–809. ISSN: 01605682, 14769360. DOI: 10.2307/2582629.

[88] J. N. Tsitsiklis. “Special cases of traveling salesman and repairman problems with time windows”. In: *Networks* 22.3 (May 1992), pp. 263–282. ISSN: 1097-0037. DOI: 10.1002/NET.3230220305.

[89] P. Turchin. *Quantitative analysis of movement: measuring and modeling population redistribution in animals and plants*. Storrs, Connecticut: Beresta Books, 2015. ISBN: 978-0-9961395-0-2.

[90] P. Vansteenwegen, W. Souffriau, and D. V. Oudheusden. “The orienteering problem: A survey”. In: *European Journal of Operational Research* 209.1 (Feb. 2011), pp. 1–10. ISSN: 0377-2217. DOI: 10.1016/j.ejor.2010.03.045.

[91] P. Vansteenwegen and D. Van Oudheusden. “The Mobile Tourist Guide: An OR Opportunity”. In: *OR Insight* 20.3 (July 2007), pp. 21–27. ISSN: 1759-0477. DOI: 10.1057/ori.2007.17.

[92] C. Verbeeck, P. Vansteenwegen, and E.-H. Aghezzaf. “An extension of the arc orienteering problem and its application to cycle trip planning”. In: *Transportation Research Part E: Logistics and Transportation Review* 68 (Aug. 2014), pp. 64–78. ISSN: 1366-5545. DOI: 10.1016/j.tre.2014.05.006.

[93] H. Wang. “A Divide-and-Conquer Algorithm for Two-Point $L_1$ Shortest Path Queries in Polygonal Domains”. en. In: *Journal of Computational Geometry* 11.1 (2020), pp. 235–282. DOI: 10.20382/jocg.v11i1a10.

[94] H. Wang. “On the geodesic centers of polygonal domains”. en. In: *Journal of Computational Geometry* 9.1 (2018), pp. 131–190. DOI: 10.20382/jocg.v9i1a5.

[95] H. Wang. “Shortest Paths Among Obstacles in the Plane Revisited”. In: *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms*. Jan. 2021, pp. 810–821. DOI: 10.1137/1.9781611976465.51.

[96] J. Wang, N. Wu, X. Lu, X. Zhao, and K. Feng. “Deep Trajectory Recovery with Fine-Grained Calibration using Kalman Filter”. In: *IEEE Transactions on Knowledge and Data Engineering* 33.3 (2019), pp. 921–934. ISSN: 2326-3865. DOI: 10.1109/TKDE.2019.2940950.

[97] S. Winter and Z.-C. Yin. “The elements of probabilistic time geography”. In: *GeoInformatica* 15.3 (Apr. 2010), pp. 417–434. ISSN: 1573-7624. DOI: 10.1007/s10707-010-0108-1.

[98] T. Xia, Y. Qi, J. Feng, F. Xu, F. Sun, D. Guo, and Y. Li. “AttnMove: History Enhanced Trajectory Recovery via Attentional Network”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 35.5 (May 2021), pp. 4494–4502. ISSN: 2159-5399. DOI: 10.1609/aaai.v35i5.16577.

# Statement of Originality

## Use of AI-Assisted Tools

For the writing of this thesis, I used AI-assisted tools like DeepLWrite and ChatGPT exclusively for grammar, style, and code improvements. Their suggestions were reviewed and thoroughly checked. I claim full responsibility for the content of this thesis.

## Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet sowie Zitate kenntlich gemacht habe.

Dortmund, den 5. August 2026
Mart Hagedoorn