Stony Brook University

OFFICIAL COPY

The official electronic file of this thesis or dissertation is maintained by the University Libraries on behalf of The Graduate School at Stony Brook University.

© All Rights Reserved by Author.

Thick Non-Crossing Paths and Minimum-Cost Continuous Flows in Polygonal Domains

A Dissertation Presented
by
Valentin Polishchuk
to
The Graduate School
in Partial fulfillment of the
Requirements
for the Degree of

Doctor of Philosophy
in
Applied Mathematics and Statistics
(Operations Research)

Stony Brook University
August 2007

Stony Brook University
The Graduate School

Valentin Polishchuk

We, the dissertation committee for the above candidate for the Doctor of Philosophy degree,
hereby recommend acceptance of this dissertation.

Dissertation Adviser Professor Joseph S. B. Mitchell
Department of Applied Mathematics and Statistics

Chairperson of Defense Professor Esther M. Arkin
Department of Applied Mathematics and Statistics

Professor Michael A. Bender
Department of Applied Mathematics and Statistics

Professor Jie Gao
Computer Science Department

This dissertation is accepted by the Graduate School

Lawrence Martin
Dean of the Graduate School

Abstract of the Dissertation

Thick Non-Crossing Paths and Minimum-Cost Continuous Flows in Polygonal Domains

by

Valentin Polishchuk

Doctor of Philosophy

in

Applied Mathematics and Statistics

(Operations Research)

Stony Brook University

2007

We study the problem of finding shortest non-crossing thick paths in a polygonal domain, where a *thick* path is the Minkowski sum of a usual (zero-thickness, or *thin*) path and a disk. Given K pairs of terminals on the boundary of a simple n-gon, we compute in $O(n + K)$ time a representation of the set of K shortest non-crossing thick paths joining the terminal pairs; using the representation, any particular path can be output in time proportional to its complexity.

In a polygonal domain with h holes we compute K shortest thick non-crossing paths in $O\left((K + 1)^h h! \operatorname{poly}(n, K)\right)$ time, using an efficient method to compute any one of the K thick paths if the "threadings" of all paths amidst the holes are specified. We show that if h is not constant, the problem is NP-hard; we also show the hardness of approximation. We give a pseudopolynomial-time algorithm for some rectilinear versions of the problem.

We apply our thick paths algorithms to obtain the first algorithmic results for the *minimum-cost continuous flow* problem — an extension of the standard discrete minimum-cost network flow problem to continuous domains. The results are based on showing a continuous analog of the Network Flow Decomposition Theorem.

We investigate the problem of finding the *maximum number* of thick paths that can be routed in a polygonal domain. Using a modification of the "continuous uppermost path" algorithm, we give a constructive proof of a continuous Menger-type result: the maximum number of paths equals to the length of a shortest path in the "thresholded critical graph" of the domain. The algorithm computes (a representation of) the paths in $O(nh + n \log n)$ time. We show how to use the algorithm to find maximum *monotone* flows and paths. For simple polygons we give faster, linear-time algorithms.

The non-crossing thick paths problems, as well as the continuous flow problems, arise in the air traffic management problem of optimally routing air traffic lanes in Flow Constrained Areas while avoiding weather hazards, no-fly zones, and other constraints. The other motivations for studying the problems come from wire routing for circuits and from information propagation in sensor networks.

We also consider other motion planning problems: the *touring* problems, shortest paths with bounded number of links in rectilinear domains, and optimal tours in "pixelated" environments and grids.

Contents

List of Figures

Acknowledgements

This dissertation could not have been completed without the guidance and support from my adviser, Professor Joseph S. B. Mitchell. His course in computational geometry initially attracted my interest to the field (and if I took another course with Joe, I would have been for sure equally fascinated with another subject). He created an exceptionally friendly research atmosphere, in which doing PhD has been a creative and fun process. I thank Joe for his understanding, both in academic and in other matters — without it finishing my PhD would have been by far harder.

It was a great pleasure to work also with Professor Esther M. Arkin and Professor Michael A. Bender from Computer Science department. Not only that I witnessed how quick thinkers solve problems, but also learnt how a course can be taught, how important collaboration is, and how one should and should not write papers.

I am thankful to Professor Alon Efrat in the University of Arizona, to Dr. Jimmy Krozel at Metron Aviation, to Professor Stephan Foldes in Tampere University of Technology, to Professor Esko Ukkonen in Helsinki Institute for Information Technology, to Jon Lenchner at IBM, to Professor Vincenzo Liberatore in Case Western Reserve University for the opportunities to visit other stimulating research environments around the world. These visits were invaluable for broadening my scientific interests.

I would like to thank the people from Applied Math and Statistics and Computer Science departments at Stony Brook, who made my PhD study a joyful experience: to professors Eugene Fienberg, Wei Zhu, Jie Gao, Steve Skiena, Alper Yildirim, and Alan Tucker, graduate students Andrei Antonenko, Eli Packer, Joondong Kim, Jason Zhou, Girishkumar Srabhani, Amithab Basu, Joe Prete and Piyush Kumar, and, of course, to the staff, Pam Wolfskill, Christine Rota, Nancy Policastro, Victor Poon, and Laura Colucci, who make everything run so smoothly.

This work is dedicated to Nina, Tatiana, and Snezhana Polishchuk. Thank you for being with me.

1 Introduction

One of the most studied subjects in computational geometry is the shortest path problem (see surveys [98, 97]): given a set of polygonal obstacles in the plane and a pair of points (s, t), find a shortest s-t path avoiding the interiors of the obstacles. Efficient algorithms are known for the two-dimensional problem both in simple polygons and in polygonal domains.

The *non-crossing paths* problem is an extension of the shortest path problem. In the problem, given a set of polygonal obstacles in the plane and K pairs of points (s_k, t_k), one wants to find a collection of K non-crossing s_k-t_k paths avoiding the interiors of the obstacles and such that the paths are optimal according to some criterion. The objective may be either to minimize the sum of the lengths of the paths (*minsum* version) or to minimize the length of the longest path (*minmax* version).

The general problem (with K being part of the input) is NP-hard even in the absence of obstacles, under the L_1 or the Euclidean metric, and under any objective — minsum or minmax [15]. On the contrary, in a *simple* polygon with all pairs (s_k, t_k) on the boundary, it is possible to build in linear time a data structure such that a shortest path can be output in time proportional to its complexity [107]. We build our solution for the case of simple polygons on the ideas from [107].

The problem of routing "thick" paths arises in a variety of applications, including VLSI, air traffic management (ATM), and robotics. A *thick* path is the Minkowski sum of a curve in $\mathbb{R}^2$ (the *reference* path) and the unit disk.

Finding *one* thick shortest path avoiding the obstacles can be done by first offsetting the obstacles by 1 and then solving the usual shortest path problem the in presence of the offset obstacles; the path found serves as the reference path for an optimal thick path [29, 89]. Thus, in a simple polygon the shortest thick path can be computed in linear time using the algorithm for shortest paths in splinegons of [44].

Multiple Thick Paths

The problem of finding *multiple* thick paths (the *Thick Non-Crossing Paths Problem*), which we consider in this thesis, is an extension of both the shortest non-crossing paths problem and the thick shortest path problem. We give rigorous formulation of the problem in Section 2. As follows from the problem name, the input to the problem consists of a set of pairs of terminals; the output — of a set of shortest non-crossing thick paths connecting the terminals.

Related Work Our problem can be viewed as a variation of the Fat-Edge Graph Drawing Problem (FEDP) [48, 49], which, in turn, is an extension of the continuous homotopic routing problem (CHRP) — a classical problem in VLSI design [32, 57, 88, 91]. A related problem is that of finding shortest paths homotopic to a given collection of paths [18, 50, 35]. The novelty of our work lies in considering the problem in simple polygons and polygonal domains; the previous research concentrated on *point* obstacles for the paths. Our data structure for storing thick shortest paths shares similar ideas with the ones used in [48, 49, 91, 107].

Remark. Although only point obstacles are considered in CHRP/FEDP, the existing results on FEDP [48, 49] are more general than ours in some other aspects: the general FEDP takes an embedding of an *arbitrary* planar graph and draws it with the edges of *maximum* thickness; we do not answer the question of finding the *maximum* separation between the paths (of course, one can simply do binary search on different path widths and use our algorithms as an oracle).

Some heuristics for finding thick non-crossing paths in polygonal domains are suggested in the VLSI literature [74], but no complexity analysis nor performance guarantees are given there. A very restricted version is considered in [6]. In a rectilinear environment, fast algorithms are known for some special cases of the minsum version [86, 119].

The disjoint paths problem is a very well known problem in graph theory. Generally, the minmax versions are harder than the minsum. In particular, it is NP-hard to approximate the number of length-bounded disjoint paths.

The geometric thick non-crossing paths problem (considered here) and the graph-theoretic disjoint paths problem are related in that thick non-crossing paths in polygonal domains correspond to disjoint paths in certain planar graphs. Throughout this work we exploit this connection in both ways: We use the hardness of the graph problem to establish the hardness of the geometric version; we also translate the geometric problem into the problem on a (path-preserving) graph.

Related to our problem is the curvature-constrained shortest path problem. The problem is NP-hard in general [111]; some special cases admit polynomial-time solutions [4, 17, 20, 21].

Flows in the Continuum

In certain applications the task is to route a large "swarm" of small objects (agents) through a polygonal environment. Finding and describing a specific path for each individual object may be an unnecessary complication. The problem then is to find an optimal *flow* (a vector field) in the domain so that any object could make its way through the domain simply by following one of the flow streamlines. The term *flow* originates from imagining an incompressible fluid flowing through the domain; optimal flow corresponds then to the best designed pipes that constrain the fluid motion.

Related Work In [77, 75, 117, 95] *maximum* flows in geometric domains were studied. Hu et al. [77, 75] used flows in discrete networks to approximate continuous flows. Strang [117] proved that the Maxflow-Mincut Theorem holds for a continuous flow; Mitchell [95] gave efficient algorithms for computing maximum flows in polygonal domains.

The study of discrete network flows is one of the most important subjects in combinatorial optimization. Besides the problem of finding *maximum* flows, the other problem attracting the most attention has been that of finding *minimum-cost* flows (aka the *transshipment* problem) [7]. The continuous counterpart of the problem asks for a geometric flow of minimum cost, where the cost is defined, e.g., as the length of the longest streamline of the flow, or as the total length of the streamlines, or as the area of the support of the flow.

Kohn and Strang studied the minimum-cost flow problem in a series of works [82, 83]. The main difference between the work of Kohn and Strang and ours is as follows. In Kohn and Strang's formulation, at *every* point on the boundary of the domain, the rate at which the flow enters the domain (i.e., the normal component of the flow) is specified. In our formulation, it is required that the flow enters/exits the domain only through the parts of the boundary designated as sources/sinks; everywhere else on the boundary the rate of the

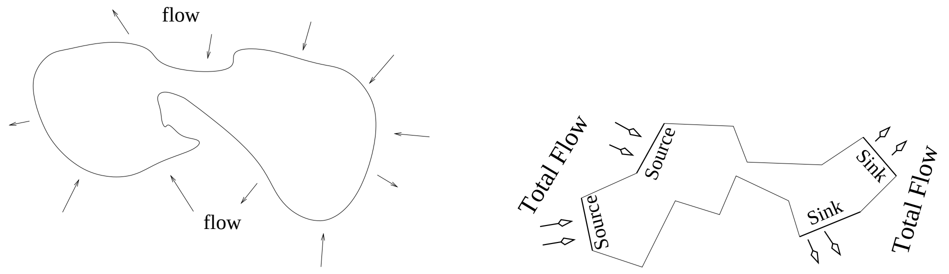


Figure 1: The difference between Kohn-Strang's formulation (left) and ours (right).

flow into/out of the domain is 0. In this way our formulation is a special case of Kohn-Strang's. On the other hand, we put a restriction only on the *integral* value of the *total* flow that comes through *all* sources/exits through *all* sinks. In this respect our formulation is more general. The difference between the formulations is schematically shown in Fig. 1.

Kohn and Strang proved that the flowlines of an optimal flow avoid unions of certain balls, and that the flowlines, corresponding to different values of the stream function, are sufficiently separated. Our main technical result, Lemma 3.2, states that thick paths, routed treating certain Minkowski sums as obstacles, do not intersect. These Minkowski sums are exactly the unions of the balls that serve as obstacles in the Kohn-Strang's construction! This is not surprising in view of our Continuous Flow Decomposition Theorem (Theorem 5.5). Our proof of Lemma 3.2 is somewhat similar to (but, possibly, more elaborate than) the proof of the analogous result in [83].[1] Moreover, we use Lemma 3.2 and its corollaries (Corollaries 3.4–3.7) to prove our Continuous Flow Decomposition Theorem. Loosely speaking, Kohn and Strang went "from flows to paths" while we go "from paths to flows".

Yet another, subtler, difference between Kohn-Strang's work and ours is the standard difference between purely mathematical and algorithmic solutions. In a purely mathematical setting the domain is not restricted to be polygonal (or, in general, have a bounded-length description); in this way purely mathematical formulation is more general than ours. On the other hand, it is often the case that for purely-mathematical solutions to exist, the input must possess certain continuity (or at least Lipschitz) properties. For instance, in [83, 116]

[1] Steinberg, Williams, and Ziemer [116] generalize the problem to higher dimensions. They also prove the result analogous to the one in [83] and Lemma 3.2; our proof is completely different from the one in [116].

it is assumed that the flow rate at the boundary of the domain is given by a Lipschitz (with respect to the geodesic distance) function. This does not have to hold in our case: the rate of the flow is 0 in any neighborhood of a source/sink; yet, the flow rate may be 1 at every point of the source/sink. This is one of the reasons why Kohn-Strang's solution cannot be used directly to solve the problem in our setting.

The last but not least difference, that, to our understanding, makes our solution a novel one, is in designing *algorithms* for actually computing the flows and in analyzing the algorithms complexity. While [83, 116] give important characterizations of the optimal flows, they do not provide a closed-form formula or another way of constructing them.

A variety of classical results and efficient algorithms exist for discrete network flows. The celebrated Maxflow-Mincut Theorem asserts that the value of the maxflow in a network equals the capacity of the mincut. The famous Flow Decomposition Theorem states that a flow can be decomposed into a set of paths from sources to sinks plus a (possibly empty) set of cycles (in particular, a *minimum-cost* flow decomposes into the paths only)[7]. Establishing the continuous versions of the Maxflow-Mincut Theorem [78, 95, 117] and the Flow Decomposition Theorem (proven here) is fundamental to exploring the mapping between discrete graph notions and their continuous analogues in geometry.

Other Work on Continuous and Geometric Flows Huge body of both theoretical and experimental work is being done in the area of numerical simulation of fluid flow, see, e.g., [63] and references thereoff.

The term "continuous flow" has been used in the literature for the flows in *discrete* networks that have *time-varying* parameters — link capacities, node storage capacities, link delays [8, 103]. In contrast, in our setting, a continuous flow is a continuous vector field (the flow) in a *static* polygonal environment (the *continuum*).

Foldes [54] showed that the permeability of Jordan curves in a domain of a steady fluid flow is a sufficient condition for the curves separability. Atkinson and Vaidya [13] and Agarwal, Efrat and Sharir [3] considered the *Euclidean transportation* problem, a generalization of the minimum-weight bipartite Euclidean matching problem, in which a commodity has to be transferred, at a minimum cost, from a set of *point* sources to a set of *point* sinks. In the *Continuous Transshipment Problem*, which we consider here, the sources and the sinks are boundary *edges* of a *simple polygon*.

Motivation

Our particular motivation for the problem comes from ATM applications in routing safe lanes ("flows") of air traffic through sectors or "flow constrained areas" (FCAs) while avoiding certain *constraints* – hazardous weather systems, no-fly zones, regions of congestion, etc. Each lane is a thick path, determined by the *protected airspace zone* (PAZ) that specifies the horizontal separation standard for flights. The routing of thick paths is critical also to the proposed "high-volume tube-shaped sector" concept for the National Airspace System in the USA (see Yousefi et. al [129]) and the related Eurocontrol "Freeway" system in Europe (see Hering [68]), which seek to lay out efficient traffic corridors, subject to geometric constraints. Using geometric max-flow techniques [95], in [101] we compute the number of lanes that can be routed through a constrained airspace. The thick non-crossing paths problem arises in optimizing the set of lanes; it is a min-cost flow problem in the continuum, as we prove in our Flow Decomposition Theorem. Algorithms based on techniques of this thesis are scheduled for implementation within the Flow-Based Route Planner (FBRP) system [84, 85, 110] at Stony Brook; FBRP utilizes greedy heuristics for routing thick paths among weather hazards, but these have no theoretical guarantees, as our algorithms here do.

Another motivation for studying thick paths and continuous flows in polygonal environments comes from data transportation in sensor networks. Suppose the data, produced at a set of sources, has to be delivered to a set of destinations by propagating through a set of sensors evenly distributed in the domain. Connecting the source-destination pairs with shortest possible paths may create undesirable congestion (Figure 2, left). One way to avoid the congestion is to introduce a capacity bound on the amount of information passing through any point of the domain. Then, a collection of optimal *well separated* source-destination paths may be sought (Figure 2, right) that provides more balanced utilization of the domain while keeping the information tracks short.

Our Contributions

In this thesis we solve the thick non-crossing shortest paths problem, which is an extension of both the thick shortest path problem and the K non-crossing shortest paths problem. We also prove the continuous Flow Decomposition Theorem and apply it to solve the *Balanced Continuous Transshipment Problem with Source-Sink Separation.*

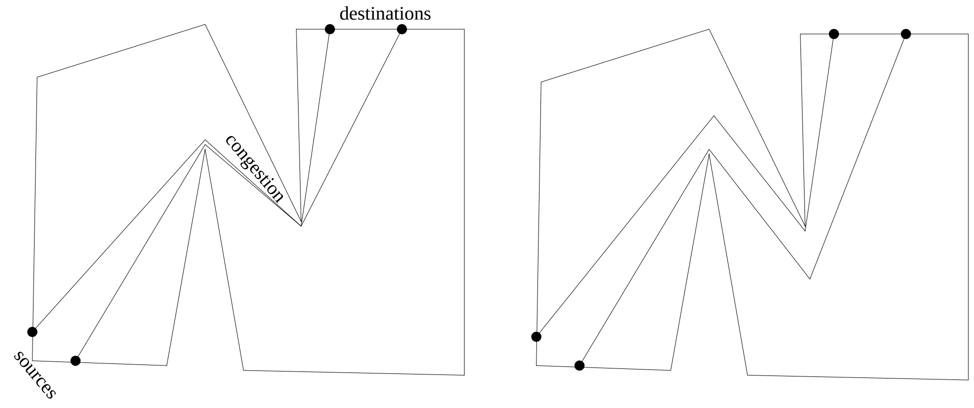


Figure 2: Well separated routes provide more balanced data transfer.

• Section 3: In $O(n+K)$ time we compute a (linear-space) representation of the set of K shortest non-crossing thick paths in a simple n-gon for a given set of K terminal pairs. The representation allows us to output the shortest thick path joining a given pair of terminals, in time proportional to the path's complexity.
• Section 4: We give an $O\left((K+1)^h h!\,\text{poly}(n,K)\right)$ algorithm for finding thick non-crossing paths in a polygonal domain with h holes. We show that if h is not constant, the minmax version of the problem is NP-hard (weakly if $K=2$, strongly for large K). We also show that unless P=NP there exists no Fully Polynomial Time Approximation Scheme for the problem, and give pseudopolynomial-time algorithms for rectilinear versions of the problem.
• Section 5: We state and prove the Flow Decomposition Theorem for minimum-cost flows in the continuum. We define the *Geometric Balanced Transshipment Problem with Source-Sink Separation*; we apply our theorem to solve the problem.
• Section 6.1: We consider rectilinear versions of the problem. We map the problem to the one on a path-preserving graph. This enables giving efficient algorithms.

Our algorithms can be used to find optimal paths under any objective that is a non-decreasing function of the length of an individual thick path; they apply, e.g., to the minsum and the minmax objectives.

2 Preliminaries

We give some definitions and introduce the notation. Let P be a simple polygon with n vertices — a simply-connected open subset of the plane whose boundary bd$P$ consists of n straight line segments disjoint other than at endpoints. In this thesis we will also use "polygon" to refer to a set whose boundary consists of straight line segments and *circular arcs*; the complexity of such a polygon is the number of its boundary segments and arcs. For any set $S \subset \mathbb{R}^2$ we speak of in the sequel, by bd$S$ we will mean the part of the boundary of S lying inside P.

A *path* π is a simple curve; $|\pi|$ denotes its length. For $r > 0$ let $\mathcal{C}_r$ be the open disk of radius r centered at the origin; we denote $\mathcal{C}_1$ with just $\mathcal{C}$. For a set $S \subset \mathbb{R}^2$ let $(S)^r$ be the Minkowski sum $S \oplus \mathcal{C}_r$; $S \oplus \mathcal{C}_r = \{x + y \,|\, x \in S, y \in \mathcal{C}_r\}$. We define a *thick path* Π within P *with reference path* π to be the Minkowski sum $\Pi = (\pi)^1$ such that Π does not intersect the exterior of P. The *length* of a thick path $\Pi = (\pi)^1$ is the length of its reference path π.

For two points v, u on the boundary of a simple polygon Q let $Q(v, u)$ denote the part of the boundary of Q from v to u clockwise. Let π be a u-v path within Q, let $B(\pi)$ be the part of Q to the right of the closed curve $v - Q(v, u) - u - \pi - v$. The points in $B(\pi)$ (resp. in $Q \setminus B(\pi)$) are said to be *below* (resp. *above*) π. If π is the *shortest* u-v path, $B(\pi)$ is called the *slice* of (v, u) and is denoted $sl(v, u)$.

Let $P^1 = P \setminus (\text{bd}P)^1$ be P offset by 1 inside. We assume that P^1 is still a simple polygon; otherwise the connected components of P^1 can be treated separately. If $\Pi = (\pi)^1$ is a thick path within P, then necessarily π is a path within P^1.

Problem Formulation Let $\mathcal{ST} = \{(s_k, t_k), k = 1 \ldots K\}$ be K pairs of points on the boundary of P^1. Borrowing terminology from the VLSI com-

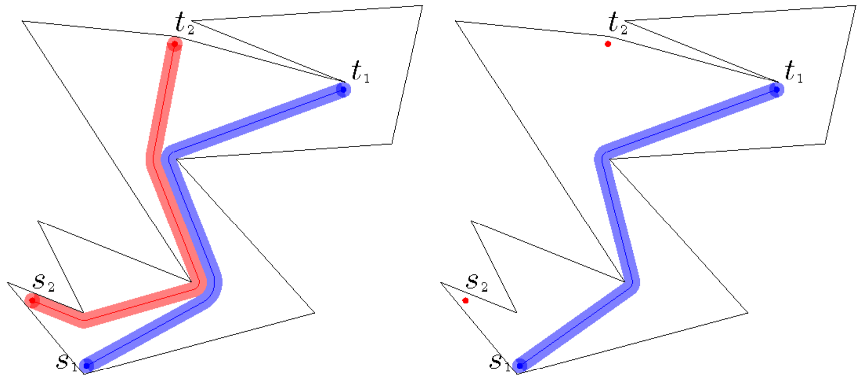


Figure 3: Left: an example. Right: a selfish thick s_1-t_1 path leaves no space for a thick s_2-t_2 path.

munity, we call the points in $\mathcal{ST}$ *terminals*. Let π_k be an s_k-t_k path within P^1; we call s_k the *start* and t_k the *destination* of the k^{th} path. Let Π_k be the thick path within P with π_k as the reference path, $\Pi_k = (\pi_k)^1$. Thick paths $\Pi_1, \ldots, \Pi_K$ are called *non-crossing* if $\Pi_i \cap \Pi_j = \emptyset$ for $i, j = 1 \ldots K, i \neq j$. Note that we allow the thick paths to share parts of the boundary with each other; we only require that the *interiors* of the paths are disjoint.

We seek a collection of thick non-crossing paths that is *simultaneously* optimal both for the minsum and the minmax version: we require that for any $k = 1 \ldots K$ the s_k-t_k path in the collection is as short as possible given the existence of (arbitrary) paths connecting the other terminals, (s_i,t_i), $i = 1, \ldots, k-1, k+1, \ldots K$. We call the collection of such paths *all-shortest*[2] [3]. See Fig. 3, left, for an example.

Following [107], we make, without loss of generality, the following assumption, which we call the *Positioning* assumption: Starting from s_1 and going around bdP^1 clockwise one encounters $s_1, s_2, \ldots, s_K$ in this order, and for any k, s_k appears before t_k. We also assume that the problem instance is feasible, i.e., that the polygon is "wide" enough to accommodate the thick paths. In particular, we assume that the distance between any two points in $\mathcal{ST}$ is at least 2 (otherwise the solution, obviously, does not exist).

The three basic problems that arise when dealing with multiple short paths

[2]Maley [91] uses the term "ideal" for a similar notion.

[3]To uniquely define the solution to the minmax version we require that each path in the collection is *locally optimal*.

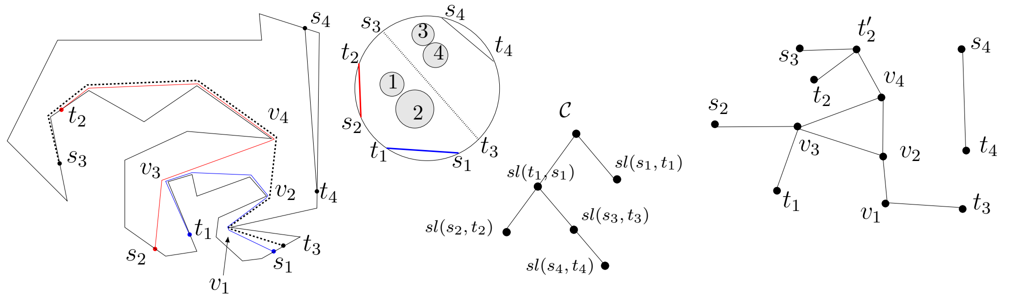


Figure 4: From left to right: an instance of the problem; the mapping of bd$P$ and the terminals to the unit circle; the tree of slices $\mathcal{T}_{sl}$; the data structure $\mathcal{G}$ storing the paths. The vertices $v_1 \dots v_4$ and t'_2 are the internal nodes of $\mathcal{G}$; t'_2 is a dummy node, added to keep t_2 a leaf.

are as follows.

(a) *Report* π_k: route one of the K thick shortest paths. In this setting we do not care how the other paths "look like". We only need the exact description of the s_k-t_k shortest path. This may be important if the paths are routed one-by-one in a distributed setting.

(b) *Store* K: build a data structure holding all K thick shortest paths. The data structure must support efficient reporting of any single path.

(c) *Report* K: Output all K paths.

Of course, (c) can be solved either by solving (a) for all $k = 1 \dots K$, or by solving (b) and then querying the data structure to report the paths one by one. Thus, in the sequel, we concentrate on solving (a) and (b).

Thin Paths We start by recollecting and extending known results on finding multiple non-crossing shortest *thin* paths [107]. For the case of thin paths we assume that the terminals in $\mathcal{ST}$ lie on the boundary of a simple polygon P. The paths are allowed to share edges and vertices, but are not allowed to cross. No path is allowed to go around a terminal of another path; in other words, the homotopy type of each path is the unique homotopy type of a path between two points on the boundary of the polygon; or, in yet another words, no path is allowed to "squeeze in" between a terminal of another path and bd$P$ — this condition will make much more sense later, when we turn to the thick paths. The paths we are looking for must be all-shortest.

The approach of [107] to the problem was as follows. First, the boundary of the polygon is mapped to the unit circle bd$\mathcal{C}$; the terminals are identified with their images. Then, a chord s_kt_k is drawn between the terminals in every pair $(s_k, t_k), k = 1 \dots K$. If two of the chords cross, then the problem instance is infeasible. Otherwise, the *tree of slices* $\mathcal{T}_{sl}$ on $\mathcal{C} \cup sl(t_1, s_1) \cup_{k=1}^{K} sl(s_k, t_k)$ is built in which the root is the whole circle $\mathcal{C}$, the root's immediate children are $sl(s_1, t_1)$ and $sl(t_1, s_1)$, and the parent-child relation is defined by containment of the slices (Fig. 4, ignore the shaded disks 1–4 now; see also [107] for details).

According to [107], the collection of the shortest paths between the pairs in $\mathcal{ST}$ forms a *forest*. In fact, the collection may not necessarily form a forest, since there may exist cycles of edges of the paths (Fig. 4); the term "forest" should be replaced with the term "graph"[4] [106]. The size of the "forest" of [107] is $O(n+K)$, and any shortest path can be reported in time proportional to the number of edges in the path. The forest is computed in a bottom-up fashion using $\mathcal{T}_{sl}$, starting in phase 1 with the paths at the leaves of the tree, and in the phase q considering the paths at level $Z-q+1$, where Z is the height of $\mathcal{T}_{sl}$. In order to achieve linear time, [107] conducts a careful refinement of the funnel paradigm for computing shortest paths in a simple polygon [65]: When a shortest s_k-t_k path at a level q is routed, the funnel from s_k in the direction of t_k is extended until the funnel hits $\mathcal{Q}$, a connected subset of the part of the boundary of the polygon, which has been already used by paths from levels $1, \dots, q-1$. The funnel then continues to extend in the direction of t_k from the other end of $\mathcal{Q}$. This leads to

Theorem 2.1. *[106, 107] In linear time a data structure can be constructed such that the shortest path between any pair of terminals can be output in time proportional to the path's complexity.*

Our solution for the case of thick paths in simple polygons (Section 3) builds and extends on ideas from [107].

[4]In Section 3.3 we build the corresponding graph for the thick paths and augment it with additional information so that it has the required features of the forest of [107]: linear size, while supporting the efficient reporting of a path.

3 Thick Paths in Simple Polygons

3.1 K=2

We first describe the solution for the case $K = 2$; the solution for arbitrary K is derived from it. We assume, without loss of generality, that s_1, s_2, t_2, t_1 appear in this order clockwise around bdP^1. Define the *bottom* B (resp. *top* T) of P^1 to be the portion of the boundary of P^1 between t_1 and s_1 (resp. s_2 and t_2): $B = P^1(t_1, s_1), T = P^1(s_2, t_2)$ (Fig. 5).

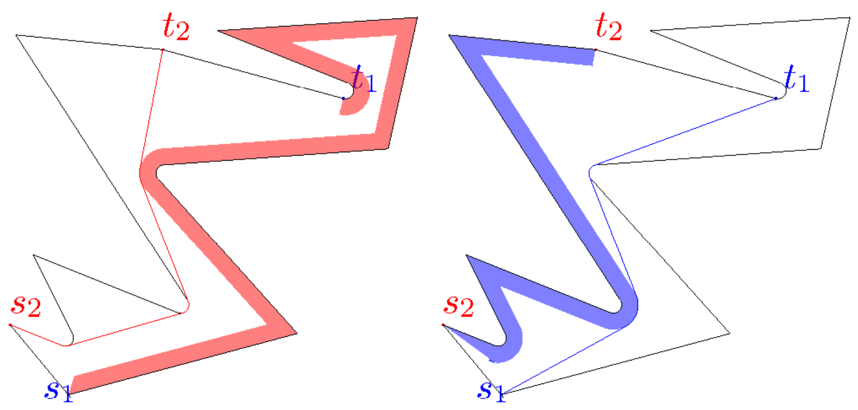


Figure 5: $B = P^1(t_1, s_1)$, $\pi_2 \cap (B)^2 = \emptyset$. $T = P^1(s_2, t_2)$, $\pi_1 \cap (T)^2 = \emptyset$. Otherwise, the other path cannot by routed.

Lemma 3.1. *π_1 cannot intersect $(T)^2$. π_2 cannot intersect $(B)^2$.*

Proof. If $\pi_1 \cap (T)^2 \neq \emptyset$, then there exists a point $p \in \pi_1$ that is within distance 2 of T, and thus there exists a point $q \in (p)^1 \subset \Pi_1$ that is within distance 1 of T and, thus, is within distance 2 of bd$P$. But the path Π_2 (of width 2) must lie between Π_1 and bd$P$, and thus any point of Π_1 must be at least at distance 2 from bd$P$. Similarly, $\pi_2 \cap (B)^2 = \emptyset$. $\square$

Let π_1^* (resp. π_2^*) be the shortest s_1-t_1 (resp. s_2-t_2) path within P^1 routed treating $(T)^2$ (resp. $(B)^2$) as obstacles (Fig. 5); let $\Pi_1^* = (\pi_1^*)^1$, $\Pi_2^* = (\pi_2^*)^1$. By Lemma 3.1, the very existence of thick s_1-t_1 and s_2-t_2 paths does not allow any s_1-t_1 path to enter $(T)^2$ and any s_2-t_2 path to enter $(B)^2$. Thus, each of Π_1^*, Π_2^* is as short as possible given the existence of the other thick path. We will now show that Π_1^* and Π_2^* are, in fact, *disjoint*, and thus, provide the solution to the thick non-crossing all-shortest paths problem.

Enclose π_1^* in an (open) "tube" τ of width 2: $\tau = (\pi_1^*)^2$. By the boundary, bdτ, of τ we will understand the boundary points of τ that lie above π_1^*. (Observe, that bdτ may be disconnected since τ may run outside P^1.) We need to prove the following (the details of the proof are deferred to the Appendix):

Lemma 3.2. *$\pi_2^* \cap \tau = \emptyset$.*

Proof. (Sketch) We consider the parts of τ induced by different parts of π_1^* one by one. We show that π_2^* "enters" and "exits" τ the same number of times. We prove that we can replace the subpath of π_2^* between the first entry and exit points by the part of bdτ and that the new path is shorter (the new path will still be feasible since it goes by the *boundary* of τ). $\square$

Theorem 3.3. *The shortest s_1-t_1 and s_2-t_2 non-crossing thick paths Π_1^* and Π_2^* can be found in linear time and space.*

Proof. $(B)^2$ and $(T)^2$, since they are an offset of P^1 by 2, are obtained by taking Minkowski sum of portions of P's boundary (simple chains) with disk of radius 3. Thus, $(B)^2$ and $(T)^2$ are found in linear time by computing the medial axis of a simple chain (part of bd$P$) [30]. We then compute π_1^*, π_2^* as shortest paths within the resulting free space splinegons, using the linear-time algorithm of [92]. $\square$

The Routing Paradigm The following corollaries can be proved similarly to the proof of Lemma 3.2, i.e., by considering the parts of π_1^*, π_2^* and π^* "pulled taught" against different features of (bd$P)^1$, (bd$P)^2$, (bd$P)^3$.

Corollary 3.4. *The paths Π_1^* and Π_2^* are all-shortest.*

Corollary 3.5. *π_1^* (resp. π_2^*) is the shortest s_1-t_1 (resp. s_2-t_2) path within P^1, routed treating $(\pi_2^*)^2$ (resp. $\tau = (\pi_1^*)^2$) as obstacle.*

Corollary 3.6. *Let $\pi^* = (bd\Pi_1^*) \cap (bd\Pi_2^*)$ be the points, boundary to both Π_1^* and Π_2^*. Then π^* is a path.*

Corollary 3.7. *Let s and t be the endpoints of π^*. Then, if the distance from each of s, t to bdP is at least 2, π^* is the* shortest *s-t path routed treating $(bdP)^2$ as obstacle.*

The corollaries assert that if Π_1^* and Π_2^* ever "meet" (so that there exist points, boundary to both), they then "go together" for some time, but after that, diverging on their way to the destinations, never meet again. While the paths go together, they do it, of course, in the optimal way.

Loosely speaking, each of the paths is routed "greedily", as opposed to "selfishly". A selfishly routed path would only care about its own length, and would "rush" to the destination in the quickest way, thus, possibly, making the routing of the other path infeasible (see Fig. 3). Our *greedy* routing assumes that each path leaves *just enough* space for the other path to "squeeze in". On the other hand, the obstacles for a path are created in a "conservative" way: no obstacle is larger than it is necessary for the existence of the other path. Thus, the paths routed greedily amidst the conservative obstacles are all-shortest.

3.2 The General Case: Arbitrary K

We give $O(n + K)$ time solutions to Problems (a) *Report* π_k and (b) *Store* K. This gives a way to solve Problem (c) *Report* K in $O(K(n + K))$ time and space, which (as we show) is worst-case optimal.

As with thin paths (Section 2), we begin by mapping bdP^1 to bd$\mathcal{C}$ and drawing the chords $s_k t_k$, $k = 1 \dots K$ (Fig. 4). Let $\mathcal{ST}^{\text{ord}} = (\nu_1, \dots, \nu_{2K})$ be the set $\{s_1, \dots, s_K, t_1, \dots, t_K\}$ ordered clockwise around bdP^1.

Definition 3.8. *Let v, u be two consecutive points in $\mathcal{ST}^{\text{ord}}$. Let γ be a path within $\mathcal{C}$ from a point on $bd\mathcal{C}(v, u)$ to a point on the chord $s_k t_k$. The k^{th}* depth *of $P^1(v, u)$, denoted $d_k(v, u)$, is defined as the minimum, over all γ, of the number of (other) chords that γ crosses.*

For example, in Fig. 4 the 4th depth of $P^1(s_2,t_2)$ is 2, the 1st depth of $P^1(t_4,t_3)$ is 1.

Let $\mathcal{O}_k$ be the set of obstacles, obtained by inflating each part of bdP^1 by 2 times its k^{th} depth (arithmetic modulo $2K$ is assumed in the indices):

$$\mathcal{O}_k = \bigcup_{j=1}^{2K} \left(P^1(\nu_j,\nu_{j+1})\right)^{2d_k(\nu_j,\nu_{j+1})} = \bigcup_{\text{edges } e\in \text{bd}P} (e)^{d'_k(e)} \qquad (1)$$

where for an edge e of bd$P$, $d'_k(e)$ denotes the amount by which the part of the boundary of P to which e belongs is inflated.

Lemma 3.9. *The sets in the right-hand side of (1) are pseudodiscs.*

Proof. Recall that we assume that the input problem instance is feasible. Suppose that two sets in (1), say $(e)^{d'_k(e)}$ and $(f)^{d'_k(f)}$, intersect. Then e and f either both lie below π_k or both lie above, since, otherwise, the problem is infeasible. Let $\Upsilon_e \subset \mathcal{ST}$ (resp. $\Upsilon_f \subset \mathcal{ST}$) be the pairs of terminals that contribute to $d'_k(e)$ (resp. $d'_k(f)$). (In terms of the mapping of bdP^1 onto bd$\mathcal{C}$, Υ_e contains the chords crossed by any path from (the image of) $(e)^1$ to the chord s_kt_k.) Let $|\Upsilon_e\cap\Upsilon_f| = z$, $|\Upsilon_e| = x+z$, $|\Upsilon_f| = y+z$, (Fig. 6, top); then, for the instance to be feasible, the distance between e and f, must be at least $2x+2y$.

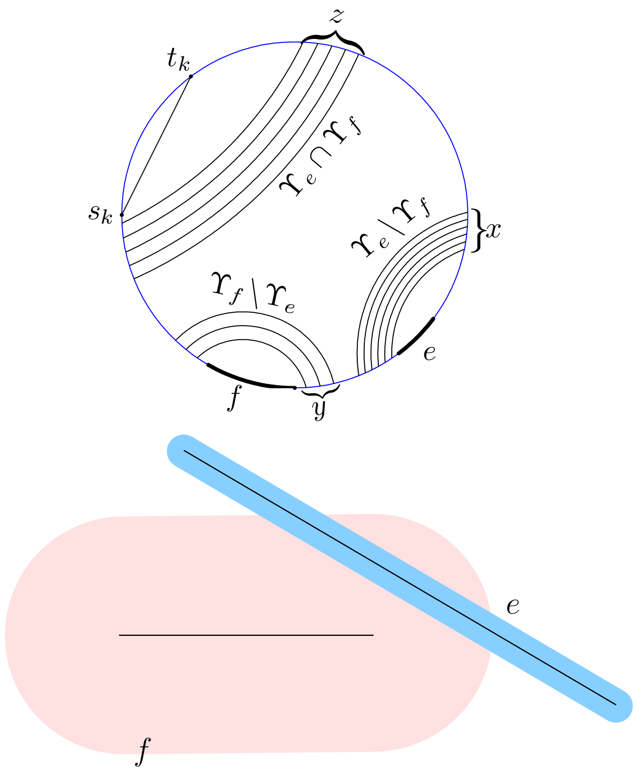
On the other hand, by our construction, $d'_k(e) = 2(x+z)+1$, $d'_k(f) = 2(y+z)+1$. If $(e)^{d'_k(e)}$ and $(f)^{d'_k(f)}$ violate the pseudodisc property, then the distance between segment e and segment f is strictly less than $|d'_k(e)-d'_k(f)| = |2x-2y|$ (see Fig. 6, bottom); since $|2x-2y| \le \max\{2x,2y\} \le 2x+2y$, we have a contradiction. □

If π^*_k, $k = 1\dots K$, is the path routed amidst $\mathcal{O}_k$ as obstacles, then each path from $\{\Pi^*_1,\dots,\Pi^*_K\} = \{(\pi^*_1)^1,\dots,(\pi^*_K)^1\}$ is as short as possible given the existence of the others. Moreover, the paths $\Pi^*_1,\dots,\Pi^*_K$ are non-crossing, by the same argument as in the proof of Lemma 3.2. Thus,

Theorem 3.10. *One of the K shortest thick non-crossing paths in a simple polygon can be found in $O(n+K)$ time and space.*

Proof. Finding the k^{th} depths of the intervals of bdP^1 can be done in $O(K)$ time.

Let n_j, $j = 1\dots 2K$ be the complexity of $P^1(\nu_j,\nu_{j+1})$; $\sum n_j = O(n+K)$. Since, by Lemma 3.9, (1) is a collection of pseudodiscs, the complexity of $\mathcal{O}_k$ is $O(\sum n_j) = O(n+K)$ [5, 81], and $\mathcal{O}_k$ can be found in $O(\sum n_j) = O(n+K)$



z
t_k
s_k
$\Upsilon_e \cap \Upsilon_f$
$\Upsilon_e \setminus \Upsilon_f$
x
$\Upsilon_f \setminus \Upsilon_e$
e
f
y
e
f

Figure 6: Top: The distance between e and f must be at least $2x+2y$. Bottom: The distance between e and f is less than $|d'_k(e) - d'_k(f)| = |2x - 2y|$.

time and space by adapting the algorithm for computing the medial axis of a simple polygon in linear time [30]. The free space for π_k^* is a splinegon and, thus, routing π_k^* amidst $\mathcal{O}_k$ can be done in linear time [92]. □

Local optimality A thick path $\Pi = (\pi)^1$ is called *locally optimal* if it cannot be shortened while staying feasible; local optimality means, in particular, that the reference path π is a sequence of straight line segments and circular arcs with the segments bi-tangent to the arcs, and that at any point, the curvature of π is at most 1. The discussion above shows that each path in a collection of shortest thick paths is locally optimal (this property will be used in Section 5):

Lemma 3.11. *For any $k = 1 \dots K$, Π_k^* is locally optimal.*

3.3 Implicit Paths Representation

The algorithm above can be run for each $k = 1 \dots K$ to get the K shortest paths in $O(K(n+K))$ time. We remark that the $O(n+K)$ space requirement of our algorithm is the *working space* requirement. The complexity of the k^{th} path may be as high as $\Omega(n+k)$ (Fig. 7), in which case the size of the output may be $\Omega(K(n+K))$, and $\Omega(K(n+K))$ *output space* may be needed just to store the paths. On the other hand, this shows that our algorithm is worst-case optimal if we require that the paths are output using explicit encoding: each path is given as the sequence of straight line segments and circular arcs.

3.3.1 The Data Structure

Alternatively, we can store the thick shortest paths in a data structure, of size $O(n + K)$, such that any path can be output in time proportional to its combinatorial complexity. Let $\mathbb{P} = \bigcup_{k=1}^{K} cl(\Pi_k)$ be the set of points that belong to (the closure of) a thick shortest path; $cl(S)$ denotes the closure of a set $S \subset \mathbb{R}^2$. Our data structure $\mathcal{G}$ is then a graph, each connected component of which corresponds to a connected component of $\mathbb{P}$ (Fig. 4).

We first describe what the *edges* of $\mathcal{G}$ are. The *nodes* of $\mathcal{G}$ correspond to those vertices of P that belong to more than one edge of $\mathcal{G}$. We use the term *nodes* for the vertices of $\mathcal{G}$ saving the term *vertices* for the vertices of P.

The reference path of any thick shortest path can be described by a sequence of vertices of P by which it goes (i.e., the vertices that cause the path to bend); the curvature (radius of the obstacle) at each vertex must also be specified. Then, to output the path in time proportional to its complexity, at

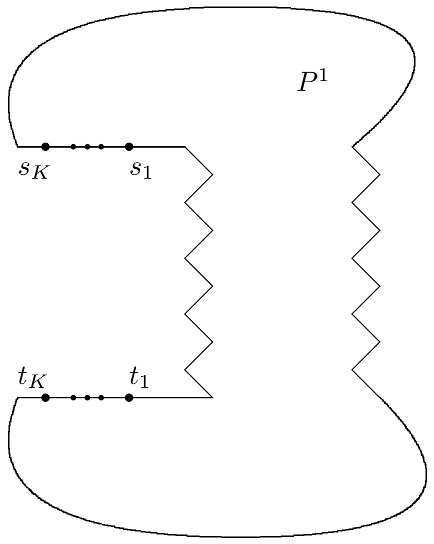


Figure 7: π_k^* can have complexity $\Omega(n+k)$.

each vertex a geometric primitive of computing a bi-tangent (or tangent, at the endpoints of the path) must be executed.

Let $\mathbb{S} = (v_{k_1}, \ldots, v_{k_r})$ be a sequence of vertices of P, consecutive along a path π_k. Suppose some other path(s) also have $\mathbb{S}$ as a subsequence of consecutive vertices. The edges of $\mathcal{G}$ are *maximal* such shared sequences of vertices; with each edge its sequence is stored. The *thickness* of an edge is the number of paths sharing it.

A node of $\mathcal{G}$ corresponds to a vertex of P at which $\mathbb{P}$ "branches", i.e., a vertex that belongs to more than one edge of $\mathcal{G}$. The leaves of $\mathcal{G}$ are the terminals $(s_k, t_k), k \in \{1 \ldots K\}$. If $\mathbb{P}$ branches at a terminal, a dummy node can be added to $\mathcal{G}$ so that the terminal and the branch point are two different nodes: the branch point is an internal node, the terminal is a leaf (like t_2 in Fig. 4).

Each internal node of $\mathcal{G}$ stores the list of incident edges sorted angularly clockwise starting with the edge closest to the boundary of P. Let $(e_1, \ldots, e_L)$ be the list of edges incident to a node u of $\mathcal{G}$. For $1 \le l \le L$, the total thickness of $e_1 \ldots e_{l-1}$ (and that of $e_{l+1} \ldots e_L$) is also stored at u, along with the edge e_l. Later, when actual routing of a path through u is performed, these thicknesses will give the curvature of the reference path at u.

Let $\Pi_k = (\pi_k)^1$ be a thick path following an edge e of $\mathcal{G}$. All the vertices in e are shared by all the paths going through it. Hence, the curvatures of π_k

on both sides of the path do not change as π_k goes along e: the (reciprocals of the) curvatures show how many other thick paths "pad" π_k on each of its sides. Thus,

Observation 1. *To route a path using $\mathcal{G}$, it is enough to know the curvatures of the path only at the endpoints of the edges of $\mathcal{G}$, i.e., at the nodes of $\mathcal{G}$.*

The other piece of information stored with each edge e is the maximal and minimal index of the paths going through the edge, $e_{\min}$ and $e_{\max}$. Since $s_1 \dots s_K$ appear in this order around $\mathrm{bd}P^1$, all paths with indexes in $[e_{\min}, e_{\max}]$ follow e; all paths with indices outside the interval do not follow the edge. To facilitate routing through $\mathcal{G}$, the pointer to each incident edge e at a node of $\mathcal{G}$ is augmented with the indexes $e_{\min}$ and $e_{\max}$; this way, when a path, routed from the start, arrives at the node, it can determine in constant time which incident edge to follow on the way to the destination.

Lemma 3.12. *The size of $\mathcal{G}$ is $O(n + K)$.*

Proof. The nodes of $\mathcal{G}$ are among the terminals in $\mathcal{ST}$ and the vertices of P. Since the paths $\Pi_1 \dots \Pi_K$ are non-crossing, $\mathcal{G}$ is planar. The total amount of information stored at a node is proportional to the degree of the node. The total amount of information stored at the edges of $\mathcal{G}$ is proportional to the size of P. □

3.3.2 Reporting a Thick Path Using $\mathcal{G}$

Routing a path π_k is done in two steps. First, we establish which edges of $\mathcal{G}$ the path follows. Since s_k is a leaf, there is a unique edge joining s_k to the rest of $\mathcal{G}$. At a node u of $\mathcal{G}$ the index of the path and the pointers to the edges incident to u tell what the next edge of the path is. This is repeated until t_k is reached and all the edges of $\mathcal{G}$ that π_k follows are known.

Second, at each node u of $\mathcal{G}$ that the path goes through, the curvature of the path is read off, and the tangent or bi-tangent is computed to extend π_k to go one more link towards t_k. Specifically, let $e = (u, v)$ be the edge that the path takes after u. The curvature κ of π_k at u can be read off the thickness of e at u. According to Observation 1, the curvature of π_k at every internal vertex of e is still κ. Let $e_1 = (v, w)$ be the next edge of $\mathcal{G}$ that π_k goes through. When the next to the last vertex of e is reached by π_k, the new curvature of the path (the curvature at v) is read off the thickness of e_1 at v, and the routing continues.

All the operations described can be performed in constant time per vertex of P along the path. Thus,

Lemma 3.13. *Given $\mathcal{G}$, a thick shortest path can be reported in time proportional to its combinatorial complexity.*

3.3.3 Computing $\mathcal{G}$

We can use the approach of [107] to actually compute $\mathcal{G}$ in linear time. The idea is to use $\mathcal{T}_{sl}$ to compute $\mathcal{G}$ in a "bottom-up" fashion. First, the paths at the bottom level of $\mathcal{T}_{sl}$ are found; this can be done in linear time by the same argument as in [107]. After the paths at level q are computed, the paths at level $q+1$ are routed in time proportional to the complexity of the part of bdP^1, not used by the paths at the levels 1 to q. The details of the algorithm are the same as in [107]: when a path π_k at the level $q+1$ is routed, the funnel from s_k in the direction of t_k is extended until the funnel hits $\mathcal{Q}$ — a connected subset of the part the boundary already used by the paths from levels 1 to q. The funnel then continues to extend in the direction of t_k from the other end of $\mathcal{Q}$.

The geometric primitives for extending the funnel — computing tangents and bi-tangents, walking through a curved trapezoidal decomposition, checking for intersections, maintaining the upper hull by "wrapping" — can still be implemented to run in constant amortized time per vertex [92]. Thus, $\mathcal{G}$, the data structure for storing thick shortest paths, can be computed within the same time bounds as the data structure of [107] for usual, thin paths.

Lemma 3.14. *$\mathcal{G}$ can be computed in $O(n+K)$ time.*

From Lemmas 3.12–3.14 follows

Theorem 3.15. *In linear time a data structure of linear size can be computed such that any path in the collection of the all-shortest thick paths within a simple polygon, can be output in time proportional to the combinatorial complexity of the path.*

4 Polygons with Holes

It was shown in [15] that the non-crossing paths problem is NP-hard: Given a set $\mathcal{ST} = (s_k, t_k)_{k=1}^{K}$ of K pairs of points in the plane, find non-crossing s_k-t_k paths, shortest under the minsum or the minmax objective function. Here, a path is allowed to go around the terminals of other paths; in this respect, every terminal can be treated as a hole (a point obstacle). The hardness of the problem stems from the fact that the terminals lie on (the boundaries of) many different holes. In contrast, if the homotopies of the paths are given, the shortest paths within the same homotopy types can be found efficiently [18, 50].

In this thesis we concentrate on the case in which all terminals lie close to the boundary of the outer polygon P. This special case is important in our motivating application (air traffic routing through a Flow Constrained Area). It is also one of the special cases considered in the study [95] of max-flows in polygonal domains, as it arises in the VLSI formulations of wire routing between terminals that lie on the boundary of none or just one of the holes [74, 119]. So, we assume that $\mathcal{ST} \subset \text{bd}P^1$ (and that the Positioning assumption holds). Of course, we continue to assume that $\forall p \in \mathcal{ST}$, $(p)^1$ does not intersect any obstacle.

As in the case of simple polygons, the algorithm developed in this section works for any objective function that is a non-decreasing function of the length of an individual path[5]. When speaking of a collection of paths, *shortest* will mean a collection that is optimal under a given objective; a *shortest path* will mean a path in an optimal collection. As in the case of simple polygons, we will ensure the uniqueness of the solution by requiring that the optimal collection is Pareto optimal: no paths can be made shorter without increasing the length

[5]This includes, in particular, the minsum and the minmax objectives. In polygonal domains, however there exist instances of our problem for which no all-shortest paths collection exists. This is in contrast with the case of simple polygons.

of another path. In particular, this means that the optimal paths are locally optimal.

We first show how to define the free space for *one* thick path, given the "threadings" of all paths and the order in which the path visits the holes; the shortest path within the free space will be a member of the optimal collection. The number of the threadings of the shortest paths is polynomial in K for fixed h (Lemma 4.1); for a large number h of holes, we show that the minmax version of the problem is weakly NP-hard if $K = 2$ and strongly NP-hard if K is the part of the input. We also prove hardness of approximation and give a pseudopolynomial-time algorithm for rectilinear versions of the problem (Section 6.1.2).

Small Number of Holes

In some applications the number of holes h may be assumed to be small. E.g., in ATM, the holes represent large weather systems and it is often the case that there are not too many of them.

We define the *threading type*, or *threading* of a path to be a vector $\chi \in \{above, below\}^h$ whose j^{th} component indicates whether the hole H_j is above or below the path. The number of different threadings of each of the shortest paths can thus be 2^h. Since, in principle, each of the K shortest paths can have any of the 2^h threadings, the total number of threadings of K paths could potentially be as high as 2^{hK}. Assuming $\mathcal{ST} \subset \text{bd}P^1$ reduces the number of different threadings substantially:

Lemma 4.1. *If the terminals lie on the boundary of P^1, the number of threading types of K shortest paths is at most $(K+1)^h$.*

Proof. K shortest paths between the pairs of terminals in $\mathcal{ST}$ cut P^1 into K+1 pieces (see Fig. 4). The threadings of the paths can be specified by indicating how the holes are put into the pieces. Since there are $K+1$ pieces and h holes, the total number of threadings of the shortest paths is at most $(K+1)^h$. $\square$

When the threadings of the K shortest paths are given, the (conservative) obstacles for each of the K paths can be constructed as follows. First, each hole is offset by 1 outside; no reference path is allowed to pass closer than 1 to any hole. Then, as in the case of simple polygons, for $k = 1 \dots K$, $j = 1 \dots h$, we define the k^{th} depth, $d_k(j)$, of the hole H_j as the minimum number of chords that a path from (the image of) H_j crosses before reaching the chord $s_k t_k$ (we assume the holes are mapped to $\mathcal{C}$ together with the pairs in $\mathcal{ST}$).

For example, the 4^{th} depth of the hole 1 in Fig. 4 is 1, the 2^{nd} depth of the hole 2 is 0, etc. Then the obstacle $\mathcal{O}_k$ for the reference path π_k^* is the union of correspondingly inflated parts of bdP^1 and boundaries of the holes:

$$\mathcal{O}_k \;=\; \bigcup_{l=1}^{2K} \big(P^1(\nu_l,\nu_{l+1})\big)^{2d_k(\nu_l,\nu_{l+1})} \quad\cup\quad \bigcup_{j=1}^{h} (\mathrm{bd}H_j)^{2d_k(j)}\,, \tag{2}$$

where we have retained the notation from Section 3. Each of the sets in (2) can be broken down to the Minkowski sums of edges of P with the disks. By an argument as in the proof of Lemma 3.9, $\mathcal{O}_k$ is the union of $O(n+K)$ pseudodiscs, and, thus, has complexity $O(n+K)$ [5, 81] and can be built in $O((n+K)\log(n+K))$ time using a randomized algorithm (see [37, Chapter 13] and [71] for further discussions and more references). We still refer to the obstacles in $\mathcal{O}_k$ as *holes* even though the original holes of the domain, when inflated according to (2), may overlap and intersect the outer boundary of the domain; in particular, the domain may become disconnected, so by *domain* we will understand the connected component that contains s_k and t_k.

After the obstacles corresponding to the given threadings are built, we need to find, for each $k = 1\dots K$, the shortest s_k-t_k path with the given threading ζ_k. Let $\Xi(\zeta_k)$ be the family of homotopy types of simple s_k-t_k paths with the threading ζ_k. Let $\Xi^*(\zeta_k) \subset \Xi(\zeta_k)$ be the family of homotopy types for which the *locally shortest* path is simple. Only one homotopy type $\chi^* \in \Xi(\zeta_k)$ can be a homotopy type of a shortest path with the threading ζ_k; clearly, $\chi^* \in \Xi^*(\zeta_k)$. (Actually, two optimal homotopy types can arise in case of degeneracies, which we will ignore without loss of generality.) We show how to enumerate all homotopy types in $\Xi^*(\zeta_k)$ by scrolling through all permutations of the holes and, for each permutation, bridging the holes to transform the domain into a simple polygon.

Let A and B be the sets of holes that are above and below (resp.) the kth path, according to ζ_k. Given an ordered sequence $\mathbb{H} = (H_1,\dots,H_h)$ of the holes and a threading ζ_k, we define the operation of *bridging* as follows: take the holes from $\mathbb{H}$ one by one and connect each hole with a shortest path to the point just above s_k (if the hole belongs to A) or just below s_k (if the hole belongs to B); treat the holes and the already-built bridges as obstacles. By construction, the bridges do not cross each other. Bridging the holes transforms the domain into a (weakly) simple polygon $P_k = P_k(\mathbb{H},\zeta_k)$.

Let π^* be the locally shortest path of homotopy χ^*, i.e., the shortest path with threading ζ_k. The following lemma shows that while scrolling through all $h!$ sequences of the holes, χ^* will be "spotted".

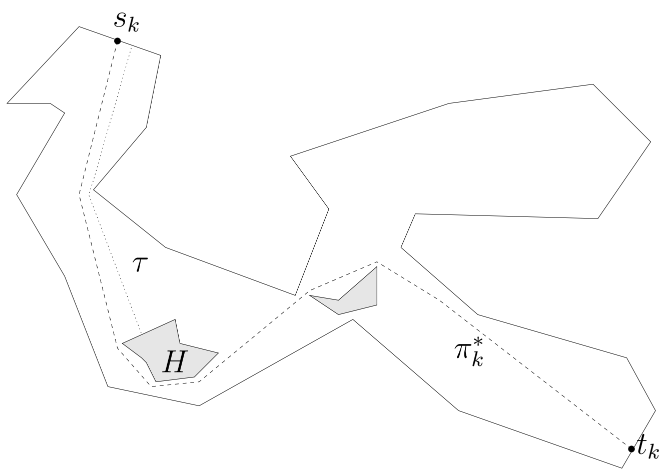


Figure 8: If $H \in A$, it is also above any s_k-t_k path in P_k. π_k^* is dashed; the bridge τ is dotted.

Lemma 4.2. *There exists an ordered sequence $\mathbb{H} = (H_1, \ldots, H_h)$ of the holes such that the homotopy type of every path in the simple polygon $P_k = P_k(\mathbb{H}, \zeta_k)$ is χ^*, where the homotopy type is viewed with respect to the original domain, without the bridges.*

Proof. By induction on the number, h^*, of holes touched by π^*. The base: If $h^* = 0$, then any sequence $\mathbb{H}$ works, since no bridge crosses π^*. Indeed, suppose the bridge τ from a hole $H' \in A$ crosses π^*. Since τ starts and ends above π^*, τ could be shortened by following π^* between the first and the last points of the crossing, contradicting the fact that τ, by construction, is the *shortest* path from H' to (just above) s_k.

The inductive step: Let H be the first hole touched by π^*; say, $H \in A$. The bridge τ routed, in the absence of any other bridges, from H to (just above) s_k does not cross π^* (otherwise, τ could be shortened). Thus, we bridge H to (just above) s_k using τ (Fig. 8), resulting in a scene with $h - 1$ holes, with every s_k-t_k path leaving H on the same side (above/below, viewed w.r.t. the original domain) as π^* leaves it. (Otherwise, the path would cross τ, which is now a part of the outer boundary.) Thus, we can start over: take the next hole touched by π^* and bridge it with a bridge not crossing π^*, and so on. $\square$

The above proof is not constructive since we do not know in advance the order in which π^* touches the holes; thus, we resort to scrolling through all possible orders. Given the threadings of the K paths, for each $k = 1 \ldots K$, we

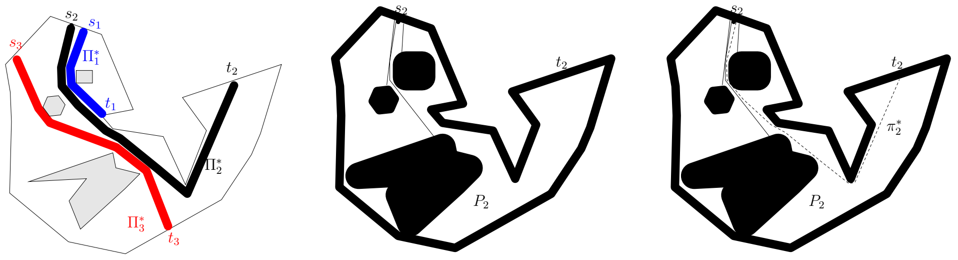


Figure 9: Left: A collection of optimal paths. Center: The free space P_2 for π_2^* after the threading and the order in which the holes are visited by the path are guessed correctly – the holes are inflated and bridged to the outer boundary. Right: The shortest s_2-t_2 path within P_2 is π_2^* (dashed).

go through all of the $h!$ permutations of the holes, bridging the holes in the order given by the permutation, and find the shortest s_k-t_k path; we keep the overall best collection of paths. The running time is $O(h!hK\tau(n+K))$ per threading.[6] Refer to Fig. 9 for an example.

Since the number of different threadings of the K shortest paths is $O((K+1)^h)$ (Lemma 4.1), we have:

Theorem 4.3. *K shortest thick non-crossing paths can be found in $O\left((K+1)^h h!hK\tau(n+K)\right)$ time.*

Remarks (1) An alternative approach to find the representative paths for every homotopy type in $\Xi^*(\zeta_k)$ is to decompose the free space into "corridors" and use the universal cover [69] lifting every corridor according to the threading ζ_k[7].
(2) Cabello et al. [27] and Bespamyatnikh [19] suggested efficient schemes for encoding the homotopy types of paths in the plane: Given a path, the algorithms in [19, 27] output its encoding in polynomial time. In our setting, a solution to the *inverse* problem is desired: given an encoding, produce a path of the corresponding homotopy type.
(3) It was shown in [50, 18] how to compute efficiently K shortest thin paths, homotopically equivalent to a given set of paths. Unfortunately, for the case

[6]We denote by $\tau(M)$ the time complexity of finding one shortest path among obstacles whose boundaries consist of a total of M straight line segments and circular arcs. We suspect that $\tau(M) = O(M \log M)$ (e.g., by extending [70]), but the current best known straightforward algorithm [29] has $\tau(M) = O(M^2 \log M)$.

[7]We thank Jack Snoeyink for this observation.

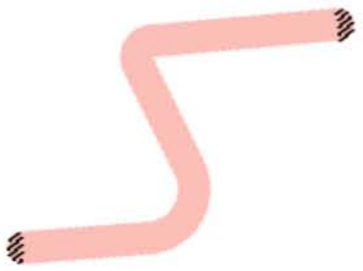


Figure 10: A thick path is a "sausage". The canonical part is what is left after the hatched semicircles are clipped off.

of thick paths, we do not have a more efficient solution than just routing the paths one by one. It is possible that using the approach of [18, 50], finding K thick shortest paths with given threadings can be done more efficiently.
(4) As in the case of simple polygons, the shortest thick paths can be stored in a linear-size data structure such that a shortest path can be reported in time proportional to its combinatorial complexity.
(5) Our algorithms generalize straightforwardly to the case in which each path has its own thickness.

Hardness Results Finding K short disjoint paths in a planar graph is NP-hard [72]. Disjoint paths in a plane graph correspond to thick non-crossing paths in a polygonal domain, created by "fattening" the graph edges. This leads to a proof of NP-hardness for the minmax version of our problem. See Section 6.1 for details.

5 Minimum-Cost Flows in the Continuum

In this section we consider the *minimum-cost continuous flow* (or, the *continuous transshipment*) problem — an extension of the standard discrete minimum-cost network flow (transshipment) problem to continuous domains. We begin by modifying slightly the statement of the thick non-crossing paths problem so that it fits into the framework of flows in the continuum [78, 95, 117]: we "clip off" the semicircular parts at the ends of thick paths and have the terminals of the paths be segments on the boundary of the polygon. This allows treating the paths as (the support of) a flow between the terminals. The fact that (the support of) a flow can be decomposed into the flow's streamlines suggests that a flow can be decomposed into (an *infinite* number of) thin paths. We exploit the idea of "gluing" thick paths (Corollary 3.6) to show that the flow can actually be decomposed into a *finite* set of *thick* paths; the size of the decomposition is linear in the size of the problem input. This is the statement of our Flow Decomposition Theorem, the continuous analogue of the famous network flow theorem. We use the theorem to reduce the continuous transshipment problem to that of finding thick non-crossing paths in the domain. The algorithms for the latter problem developed in the previous sections allow us to solve a class of instances of the transshipment problem efficiently.

Canonical Part of a Thick Path For two points a, b let $\frac{\mathcal{C}}{2}(a, b)$ be the (open) semicircle, with diameter ab, to the left of the segment ab. Let $\Pi = (\pi)^1$ be a thick s-t path. Let $s's''$ (resp. $t't''$) be the diameter of $(s)^1$ (resp. $(t)^1$), perpendicular to π at s (resp. at t); let s', t' (resp. s'', t'') lie below (resp. above) π. We define the *canonical part*, $\Pi^{\square}$, of Π to be its part between $s's''$ and $t't''$: $\Pi^{\square} = \Pi \setminus \frac{\mathcal{C}}{2}(s', s'') \setminus \frac{\mathcal{C}}{2}(t'', t')$ (Fig. 10).

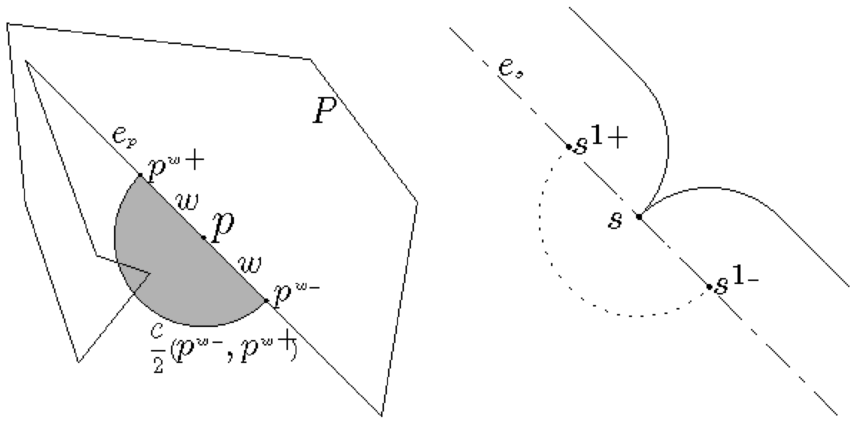


Figure 11: Left: $\mathcal{C}/2(p^{w-}, p^{w+}) \in \mathcal{R}_p$ (shaded) is attached to P along $p^{w-}p^{w+}$. Right: A thick path originating at s will be perpendicular to e_s. bd$P$, bd$\mathcal{P}$ and bd$\mathcal{P}^1$ are shown with dashed, dotted and solid lines respectively.

For a point p on bd$P$ let e_p be the edge of P on which p resides. For $w \in \mathbb{R}^+$ we say that p is *w-inside* e_p if the distance from p to the endpoints of e_p is greater than w. For a point p that is w-inside e_p, let $p^{w-} \in e_p$ (resp. $p^{w+} \in e_p$) be the point at distance w from p going counterclockwise (resp. clockwise) along bd$P$: $\{p^{w-}, p^{w+}\} = e_p \cap \mathcal{C}(p, w)$. As in [95], we augment P with a Riemann sheet $\mathcal{R}_p$, attached to P along $p^{w-}p^{w+}$ and place the semicircle $\frac{\mathcal{C}}{2}(p^{w-}, p^{w+})$ in $\mathcal{R}_p$. (The reason for placing $\frac{\mathcal{C}}{2}(p^{w-}, p^{w+})$ in a separate sheet $\mathcal{R}_p$ and not in $\mathbb{R}^2$, the base sheet where P lives, is to make sure that $\frac{\mathcal{C}}{2}(p^{w-}, p^{w+})$ does not intersect P even if they overlap.) Refer to Fig. 11, right.

Lemma 5.1. *Let the points $s, t \in bdP$, such that $|st| > 2$, be 1-inside e_s and e_t respectively. Let $\mathcal{P}_{st} = P \cup \frac{\mathcal{C}}{2}(s^{1-}, s^{1+}) \cup \frac{\mathcal{C}}{2}(t^{1+}, t^{1-})$ be P augmented with the two semicircles that reside in the corresponding sheets, $\mathcal{R}_s$ and $\mathcal{R}_t$ (so that $\mathcal{P}_{st} \subset \mathbb{R}^2 \cup \mathcal{R}_s \cup \mathcal{R}_t$ is still a simple polygon). Let $\Pi = (\pi)^1$ be a thick s-t path within $\mathcal{P}_{st}$. Then π is perpendicular to e_s at s and to e_t at t.*

Proof. Let $\mathcal{P}^1 = \mathcal{P}_{st} \setminus (\text{bd}\mathcal{P}_{st})^1$ be the inward offset by 1 of $\mathcal{P}_{st}$. Since $\Pi = (\pi)^1$ is a thick path within $\mathcal{P}_{st}$, π is a path within $\mathcal{P}^1$. Locally at s, the boundary of $\mathcal{P}^1$ consists of two quarter-circles (parts of bd$(s^{1-})^1$ and bd$(s^{1+})^1$), which meet at s normally to e_s (Fig. 11). Thus, any path within $\mathcal{P}$ originating at s is normal to e_s at s. The same argument works for π at t. $\square$

Stick Representation of a Thick Path By Lemma 3.11, the canonical part of a shortest thick path $\Pi = (\pi)^1$ can be written as $\Pi^\square = \cup_{x\in\pi} n_x$, where n_x is a segment of length 2 centered at $x \in \pi$ and perpendicular to π at x. Imagine that n_x is a stick whose center, x, moves along π. Then as x moves from s to t, n_x sweeps $\Pi^\square$.

In what follows, by a *w-thick path* $\Pi = (\pi)^w$ we will mean the canonical part of $(\pi)^w$; as before, a *thick path* means (the canonical part of) a 1-thick path. For $\vec{w} = (w_1, \ldots, w_K) \in \mathbb{R}^{+K}$, disjoint thick paths $\Pi_1 \ldots \Pi_K = (\pi_1)^{w_1} \ldots (\pi_K)^{w_K}$ will be called a collection of *$\vec{w}$-thick* paths.

"Gluing" Thick Paths Let the origins of a collection of shortest thick non-crossing paths be spaced along a segment so that the distance between neighboring origins is equal to the sum of the corresponding path widths; let the destinations of the paths be spaced similarly on another segment. Then routing the paths can be reduced to routing just one thick path from the "midpoint" of the origins to the "midpoint" of the destinations (Fig. 12):

Lemma 5.2. *Let points $s, t \in bdP$ be 2-inside e_s, e_t; let $|st| > 4$. Let $s_1 = s^{1-}$, $s_2 = s^{1+}$, $t_2 = t^{1-}$, $t_1 = t^{1+}$. (By triangle inequality, $|s_1t_1|, |s_2t_2| > 2$.) Let $\Pi_1^*, \Pi_2^* = (\pi_1^*)^1, (\pi_2^*)^1$ be the two thick shortest paths between (s_1, t_1) and (s_2, t_2), let $\Pi^* = (\pi^*)^2$ be the shortest 2-thick s-t path within P. Then $cl(\Pi^*) = cl(\Pi_1^* \cup \Pi_2^*)$.*

Proof. Let $\pi = (\text{bd}\Pi_1^*) \cap (\text{bd}\Pi_2^*)$ be the points, boundary to both Π_1^* and Π_2^*. Observe that $s, t \in \pi$. By Corollary 3.6, π is an s-t path. Since $|st| > 4$, by Corollary 3.7, π is the shortest 2-thick s-t path within P, i.e., $\pi = \pi^*$. The lemma follows now from the stick representations of Π_1^*, Π_2^* and Π^*. $\square$

Lemma 5.2 generalizes straightforwardly to an arbitrary number of paths of arbitrary thicknesses.

The Segment Interconnection Problem Let $\mathcal{I}_\mathcal{S} = \cup_1^K s_k^{w_k-} s_k^{w_k+}$, $\mathcal{I}_\mathcal{T} = \cup_1^K t_k^{w_k+} t_k^{w_k-}$ be two collections of K segments each; s_k, t_k are the midpoints of $s_k^{w_k-} s_k^{w_k+}, t_k^{w_k+} t_k^{w_k-}$. Let $\mathcal{I}_{\mathcal{ST}} = \mathcal{I}_\mathcal{S} \cup \mathcal{I}_\mathcal{T}$, $\mathcal{ST} = \cup_1^K (s_k, t_k)$. We call both the segments in $\mathcal{I}_{\mathcal{ST}}$ and the points in $\mathcal{ST}$ *terminals.* Let the following properties hold: (1) every terminal is on the boundary of P; (2) s_k is w_k-inside e_{s_k}; (3) t_k is w_k-inside e_{t_k}; (4) the terminals in $\mathcal{ST}$ are sufficiently separated: $\forall k\, |s_kt_k| > 2w_k$, and $\forall p \in \{s_i, t_i\}$, $\forall q \in \{s_j, t_j\}, i \neq j$, $|pq| > w_i + w_j$: (5) the Positioning assumption (Section 2) holds, i.e., $s_1^{w_1-}, s_1, s_1^{w_1+}, \ldots, s_K^{w_K-}, s_K, s_K^{w_K+}$ appear in this order clockwise around bd$P$ and $\forall k, s_k^{w_k-}, s_k, s_k^{w_k+}$ appear before

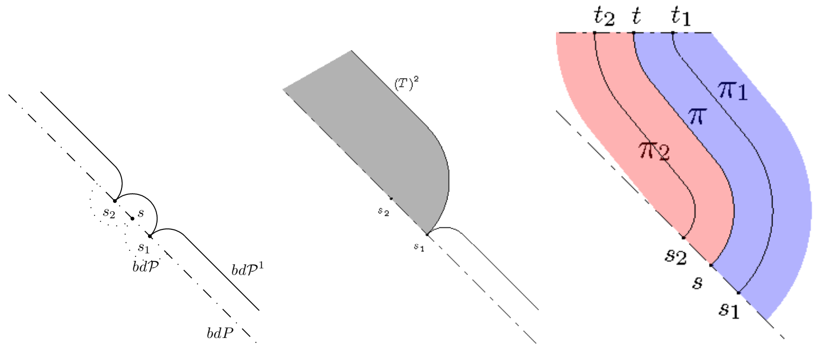


Figure 12: Left: $\mathcal{P} = P \cup \frac{\mathcal{C}}{2}(s_1^{1-}, s_1^{1+}) \cup \frac{\mathcal{C}}{2}(s_2^{1-}, s_2^{1+})$. bd$P$, bd$\mathcal{P}$ and bd$\mathcal{P}^1$ are shown with dashed, dotted and solid lines respectively. Center: recall (Section 3) that π_1 is routed within $\mathcal{P}^1$ treating $(T)^2 = (\mathcal{P}^1(s_2, t_2))^2$ as obstacles. Right: $cl((\pi_1)^1 \cup (\pi_2)^1) = cl((\pi)^2)$.

t_k^{w+}, t_k, t_k^{w-}.[8] Consider the *Segment Interconnection Problem* (SIP) — the problem of connecting $s_k^{w_k-}s_k^{w_k+}$ to $t_k^{w_k+}t_k^{w_k-}$ by K non-overlapping thick "strips" of thicknesses $w_1 \dots w_K$. By Lemma 5.1, the SIP can be formally stated as the problem of finding (the canonical parts of the) shortest thick non-crossing s_k-t_k paths within $\mathcal{P}_{\mathcal{ST}} = P \bigcup_1^K \left(\frac{\mathcal{C}}{2}(s_k^{w_k-}, s_k^{w_k+}) \cup \frac{\mathcal{C}}{2}(t_k^{w_k+}, t_k^{w_k-})\right)$, i.e., P augmented with the semicircles, lying in the corresponding Riemann sheets.

Let SIP$(P, K, \vec{w}, \mathcal{I}_\mathcal{S}, \mathcal{I}_\mathcal{T})$ be the segment interconnection problem as defined above. Let SIP$^\Sigma(\cdot)$ and SIP$^{\max}(\cdot)$ be optimal solutions to the minsum and minmax versions of the SIP$(\cdot)$.

Theorem 5.3. SIP$^\Sigma(\cdot)$ = SIP$^{\max}(\cdot)$*, and their representation can be found in* $O(n + K)$ *time.*

Proof. It follows from our results in Section 3 that there exists a collection of *all-shortest* $\vec{w}$-thick non-crossing (s_k-t_k) paths within $\mathcal{P}_{\mathcal{ST}}$ and that (a representation of) it can be found in linear time by "inflating" the parts of the boundary of the polygon appropriately. By Lemma 5.1, the paths give the solution to the SIP$(\cdot)$. □

[8]For the separation property (4) it is enough to require that the terminals are separated only in terms of the *geodesic* distance within P.

Remark Of course, there may be multiple optimal solutions to $\mathrm{SIP}^{\max}(\cdot)$. The statement of Theorem 5.3 must be understood in the sense that there exists a solution that is (Pareto) optimal for both versions.

Flows in Networks Recall some basic facts about network flows [7]. A *capacitated network* $N = (V, E)$ is a directed graph on a set of nodes V and a set of edges E, such that each edge $e \in E$ has associated with it two (non-negative) integers u_e and c_e, called the *capacity* and the *cost* of the edge. An *s-t flow* in N is a function $f : E \mapsto \mathbb{Z}_0^+$ such that $\forall i \in V \setminus s \setminus t$, $\sum_{(i,j)\in E} f((i,j)) = \sum_{(j,i)\in E} f((j,i))$ and $\forall e \in E, f(e) \le u_e$. The *value* of the flow $V = \sum_{(i,t)\in E} f(i,t)$. The *cost* of the flow is $\sum_{e\in E} f(e)c(e)$. In the *maximum* flow problem, a flow, maximizing V is sought; in the *minimum-cost* flow, V is given and the goal is to minimize the cost.

An *s-t cut* in N is the partition of V into disjoint sets S and T, $N = S \cup T$, $S \cap T = \emptyset$, such that $s \in S$, $t \in T$. An edge $e \in E$ connecting the vertices i and j of V is said to *cross* the cut if $i \in S$, $j \in T$ or $i \in T$, $j \in S$. The *capacity* of the cut is the sum of the capacities of all edges that cross it. The *minimum cut*, or *mincut* through the network is a cut of minimum capacity. The Maxflow-Mincut Theorem states that the value of the maximum *s-t* flow equals to the capacity of the mincut through the network.

The Flow Decomposition Theorem states that any flow can be decomposed into a set of *s-t* paths and a set of cycles. The set of cycles in the decomposition is empty for a minimum-cost flow.

The mincut, maxflow and min-cost floe can be computed efficiently [7].

Continuous Transshipment Problem We recollect from [95, 117] some notions related to flows in polygonal domains. Let $\Gamma_s = \{I_1 \ldots I_S\}$, $\Gamma_t = \{O_1 \ldots O_T\}$ be two sets of disjoint (open) segments on bd$P$. A *flow* in P is a vector field $\sigma : P \mapsto \mathbb{R}^2$ such that: (1) $\forall x \in P$, $\operatorname{div} \sigma(x) = 0$, i.e., there is no sources/sinks inside P; (2) $\forall x \in \mathrm{bd}P \setminus \Gamma_s \setminus \Gamma_t$, $\sigma(x) \cdot \mathbf{n}(x) = 0$, where $\mathbf{n}(x)$ is the outward pointing unit vector normal to bd$P$ at x, i.e., the flow penetrates bd$P$ only at $\Gamma_s \cup \Gamma_t$; (3) $\forall x \in P$, $|\sigma(x)| \le 1$, i.e., each point in P has *unit capacity*. The *value* of the flow is defined as $V = \int_{\Gamma_t} \sigma \cdot \mathbf{n}\, ds$. The *maximum flow* problem [78, 95, 117] asks for a flow, maximizing V; in the *minimum-cost flow* (or, the *transshipment*) problem (considered here), V is given and the flow, minimizing certain objective function, is sought[9].

[9]The classical divergence theorem states that $\forall \sigma$, $\int_{\Gamma_t} \sigma \cdot \mathbf{n}\, ds \le |\gamma^*|$, where $|\gamma^*|$ is the length of the *mincut* through the domain [117, 95]. We will assume that in our problem

The Objective Functions Let $\Gamma_s^* \subset \cup_1^S I_i$ be the part of bd$P$ through which σ actually enters P: $\Gamma_s^* = \{x \in \cup_1^S I_i \,|\, \sigma(x) \cdot \mathbf{n}(\mathbf{x}) \neq 0\}$. Define Γ_t^* similarly. For $s \in \Gamma_s^*$ let $\ell_s = \cup_{\tau=0}^{\infty} \left(s + \int_0^\tau \sigma \, dt\right)$ be the *streakline* of σ, going through s; since we only consider steady flows, it coincides with the *streamline* — curve, at every point of which σ is tangent to the curve, and with the *pathline* — the path taken by a particle released at s in the velocity field given by σ. Let $|\ell_s|$ denote the length of ℓ_s.

Strang [117] and Mitchell [95] suggested that the *cost* of a flow may be defined as: (I) the area of the support, supp σ, of the flow, where supp $\sigma = \{x \in P : |\sigma(x)| > 0\}$; or, (II) the "total length of the streamlines", $\int_{s \in \Gamma_s^*} |\ell_s| ds$; or, (III) the length of the longest streamline $\max_{s \in \Gamma_s^*} |\ell_s|$.

It is not hard to see that it suffices to consider only those flows σ that satisfy the following properties: (i) the absolute value of the flow vector is everywhere at its upper bound 1, $\forall x \in$ supp σ, $|\sigma(x)| = 1$; (ii) Γ_s^* and Γ_t^* each consists of at most $S+T$ connected components; (iii) there are no closed streamlines in σ; (iv) σ leaves Γ_s^* (and enters Γ_t^*) normally to bd$P$, $\forall x \in \Gamma_s^* \cup \Gamma_t^*$, $|\sigma(x) \cdot \mathbf{n}(x)| = 1$; (v) the lengths $|\Gamma_s^*|$ and $|\Gamma_t^*|$ of Γ_s^* and Γ_t^* satisfy $|\Gamma_s^*| = |\Gamma_t^*| = V$. (Of course, (i)–(v) are not independent, but we do not care.)

Flow Decomposition Theorem For $s \in \Gamma_s^*$ let $d(s) \in \Gamma_t^*$ denote "the other end" of the streamline ℓ_s (the destination of a particle released at s), $d(s) = s + \int_0^\infty \sigma \, dt$. Let $o(\cdot)$ be the "inverse" of $d(\cdot)$, i.e., for $t \in \Gamma_t^*$ let $o(t) \in \Gamma_s^*$ be the "origin" of a particle that ended up at t: $o(t) = s | d(s) = t$.

For the flows, enjoying the Properties (i)–(v) above, $d(\cdot)$ is continuous almost everywhere on Γ_s^*. (In the intervals of continuity of $d(\cdot)$ the Fréchet distance $\mathcal{F}(\ell_a, \ell_b)$ between the streamlines ℓ_a and ℓ_b and the distance $|d(a)d(b)|$ between $d(a)$ and $d(b)$ satisfy $\mathcal{F}(\ell_a, \ell_b) = |d(a)d(b)| = |ab|$.) Similarly, $o(\cdot)$ is continuous almost everywhere on Γ_t^*. Let $\mathbb{O} = \{o_1 \dots o_{S^*}\} \subset \Gamma_s^*$ and $\mathbb{D} = \{d_1 \dots d_{T^*}\} \subset \Gamma_t^*$ be the points of discontinuity of $d(\cdot)$ and $o(\cdot)$. Let $\mathcal{I}_S$ (resp. $\mathcal{I}_T$) be the set of segments into which Γ_s^* (resp. Γ_t^*) is decomposed by $o_1 \dots o_{S^*}$ (resp. $d_1 \dots d_{T^*}$); these segments are the maximal contiguous subsets of Γ_s^*, Γ_t^* such that the flow from a subset of Γ_s^* goes to a subset of Γ_t^*. By definition, the number of segments in $\mathcal{I}_S$ equals the number of segments in $\mathcal{I}_T$; call this number K.

The support of the flow can be "slit" along the streamlines originating at o_i^-, o_i^+, $i = 1 \dots S^*$ and the streamlines ending at d_j^-, d_j^+, $j = 1 \dots T^*$ into K "smaller" flows going from the segments in $\mathcal{I}_S$ to the corresponding segments in

$V \leq \gamma^*$, i.e., that the problem is feasible.

$\mathcal{I}_T$. Formally, consider segments $e \in \mathcal{I}_S$ and $f = d(e) \in \mathcal{I}_T$. ($|e| = |f|$.) Let σ_e be the restriction of σ on the subset of P through which the streamlines from e to f pass: supp $\sigma_e = \{x \in P \,|\, \exists s \in e | x \in \ell(s)\}$, $\sigma_e = \sigma$ on supp σ_e, $\sigma_e = 0$ o.w. Then $\sigma = \sum_{e \in \mathcal{I}_S} \sigma_e$, i.e., a flow can be decomposed into "sub-flows", such that each sub-flow goes from a single source segment to a single sink segment.

Let $ab \in \mathcal{I}_S$, $cd \in \mathcal{I}_T$ be two segments such that the (sub-)flow σ_{ab} of value $2w$, $w \in \mathbb{R}$, goes from ab to cd: $\forall x \in ab, d(x) \in cd$, $\forall y \in cd, o(y) \in ab$; by Properties (iv) and (v), $|ab| = |cd| = 2w$. Let π_{ad} and π_{bc} be the streamlines going from a to d and from b to c. For $\xi \in (0,1)$ let $\pi(\xi)$ be the (thin, usual) path from $ab(\xi) = (1-\xi)a + \xi b$ to $cd(\xi) = (1-\xi)d + \xi c$, routed treating $(\pi_{ad})^{2w\xi}$ and $(\pi_{bc})^{2w(1-\xi)}$ as obstacles. Then $\pi(\xi)$, the set of points at distance $2w\xi$ from π_{ab} and at distance $2w(1-\xi)$ from π_{cd} (so that the Fréchet distance between $\pi(\xi)$ and π_{ab}, $\mathcal{F}(\pi(\xi), \pi_{ab}) = 2w\xi$, $\mathcal{F}(\pi(\xi), \pi_{cd}) = 2w(1-\xi)$), is the streamline of σ going from $ab(\xi)$ to $cd(\xi)$. Since exactly one streamline passes through each point of the flow support, the support can be written as the union of the streamlines: supp $\sigma_{ab} = \cup_{\xi \in (0,1)} \pi(\xi)$. Lemma 5.4 below proves that supp σ_{ab} is in fact a shortest w-thick s-t path within P, where $s = ab(\frac{1}{2})$ and $t = cd(\frac{1}{2})$ are the midpoints of ab and cd.

Lemma 5.4. *supp σ_{ab} is a w-thick shortest path from s to t.*

Proof. For $M \in \mathbb{N}$ let ab^M, cd^M be the subdivisions of ab and cd into M segments of length $\delta = 2w/M$. Consider the Segment Interconnection Problem SIP$(P, M, \delta\vec{1}^M, ab^M, cd^M)$, where $\vec{1}^M \in \mathbb{R}^M$ is the vector of M ones. By Theorem 5.3, the solution to the SIP is given by a set of M $\frac{\delta}{2}$-thick shortest paths between the mid-points of the segments in ab^M and cd^M. By Lemma 5.2, these paths can be "glued" into one w-thick shortest s-t path. Since the above is true for arbitrary M, it is also true in the limit, $M \to \infty$, when the paths become the streamlines of the flow. □

Thus, the discrete network Flow Decomposition Theorem can be extended to the continuum:

Theorem 5.5. Continuous Flow Decomposition Theorem (CFDT) *The support of a minimum-cost flow can be decomposed into a set of thick paths; the size of the decomposition is linear in the size of the description of the flow.*

See Fig. 13 for an example.

The celebrated Network Flow Decomposition Theorem states that any flow can be decomposed into a set of paths from the sources to sinks and a set of

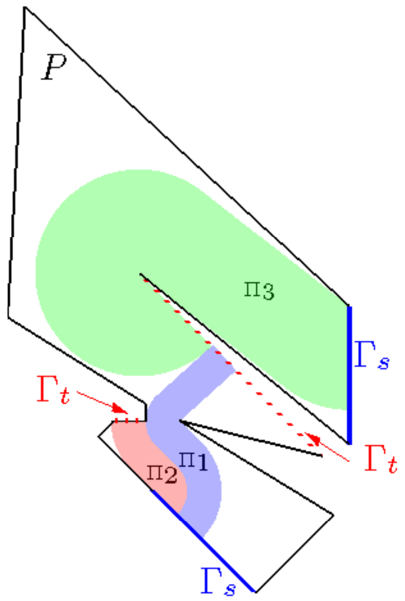


Figure 13: Flow σ from Γ_s (solid blue segments) to Γ_t (dotted red segments). supp $\sigma = \Pi_1 \cup \Pi_2 \cup \Pi_3$.

cycles. For a *minimum-cost* flow the set of cycles is empty; it is exactly for this reason that the minsum version of the famous disjoint paths problem can be formulated as a flow problem. Theorem 5.5 can be regarded as a natural extension of the Network Flow Decomposition Theorem to the continuum.

Balanced Transshipment with Source-Sink Separation The above discussion gives a solution to the transshipment problem, restricted as follows. First, assume that the problem is *balanced*, i.e., $|I_1| + \ldots + |I_S| = |O_1| + \ldots + |O_T| = V$ ("supply"="demand"). In this case $\Gamma_s^* = \Gamma_s$, $\Gamma_t^* = \Gamma_t$. Assume also that the segments in Γ_s do not "interleave" with those in Γ_t around bd$P$, i.e., that going around bd$P$ one encounters all sources and then all sinks (this is often the case, e.g., in ATM: the air traffic is routed through a sector mostly in one direction, say, East-to-West or West-to-East). Then the sets $\mathcal{I}_S$ and $\mathcal{I}_T$ can be deduced by a simple matching procedure. Let $p \in$ bd$P$ be a point between Γ_s and Γ_t, let $e \in \Gamma_s$, $f \in \Gamma_t$ be the segments, closest to p around bd$P$; without loss of generality, $|e| \leq |f|$. Let f' be the part of f, closest to p, such that $|e| = |f'|$. Insert e (resp. f') into $\mathcal{I}_T$ (resp. $\mathcal{I}_T$), delete e (resp. f') from Γ_s (resp. Γ_t), and repeat.

Since exactly one streamline of σ goes through each point of supp σ, the objective functions (I) and (II) (see page 32) are easily seen to be equivalent. By Theorem 5.3, they are also equivalent to (III). By Theorem 5.5 the transshipment problem can be reduced to the corresponding instance of the SIP. Thus, by Theorem 5.3:

Theorem 5.6. *The balanced transshipment problem with the source/sink separation as described above can be solved in linear time and space.*

Remarks (1) The Flow Decomposition Theorem applies also to the case in which the sources and the sinks belong to the outer boundary of a polygonal domain with h holes. The number of paths in this case is at most $S + T + h$. To see this, decompose the flow into two parts: from Γ_s^* to γ^* and from γ^* to Γ_t^*, where γ^* is the mincut through the domain. The bound on the number of paths follows from the fact that γ^* has at most $h + 1$ connected components.
(2) In Section 3.3 we built a data structure for storing thick shortest paths; it is an extension of the data structure of Papadopolou [107] for storing *thin* paths (called in [107] the "forest" of shortest paths). The flows in the continuum provide a new view on these data structures: it can be seen that our data structure actually represents the support of the min-cost flow from $\cup_1^K(s_i^{1+}, s_i^{1-})$ to $\cup_1^K(t_i^{1-}, t_i^{1+})$. It is worth noting that the *maximum* flows through a domain [95] can indeed be stored in a forest (the appropriately defined skeleton of the domain), while our data structure $\mathcal{G}$ (and, in fact, the data structure of Papadopolou, [106]) may have to be a general planar graph. This emphasizes the difference between maximum and minimum-cost flows.
(3) We presented two ways of specifying an instance of the thick shortest paths problem. In the first specification (Section 2) the terminals of the paths are a set of points close to bd$P$; the paths sought are the Minkowski sums of thin paths with the disks. In the second the terminals are segments on bd$P$ and the paths are the canonical parts of the Minkowski sums. From a distant view the two specifications look the same: in the second one we just have the paths "protrude" through bd$P$ so that paths' non-canonical parts (the semicircles) reside on the corresponding Riemann sheets. Although, as shown in Section 5, the two specifications are equivalent, we decided to present both since we feel that the first one reflects the standard view on the thick paths as the Minkowski sums, while the second allows one to think of the terminals as the "doors" through which the flow of agents enters the domain.

6 Variations

In this section we look at our problems in *rectilinear* domains. We also consider routing multiple *thin* paths in polygonal domains with holes. Finally, we investigate the problems of routing *maximum number* of thick paths and finding *monotone* flows and paths—direct extension of the *maximum continuous flow* problem [117, 95].

6.1 Thick Paths and Flows in Rectilinear Domains

Here we study thick *rectilinear* paths and *rectilinear* minimum-cost flow in *rectilinear* domains. We will assume that a thick path is the Minkowski sum of the reference path and the unit square. The notation $(S)^1$ will consequently mean the Minkowski sum of a set $S \subset \mathbb{R}^2$ and the unit square.

We first prove that the minmax version of the thick non-crossing paths problem is NP-hard; even if $K = 2$ and the paths are monotone, it is weakly NP-hard. Our hardness proofs, in fact, work for the Euclidean version just as well. We argue that unless P=NP there exists no Fullypolynomial-Time Approximation Scheme for the problem. For the cases $K = 2, 3$ we suggest a pseudopolynomial-time algorithm.

We then consider the rectilinear version of the minimum-cost flow problem. We show, by a standard "snapping" argument, that the problem can be reduced to the flow problem on a path-preserving graph. It allows, under certain restrictions on the placement of the terminals, to solve the minsum version of the problem of finding thick non-crossing rectilinear paths in rectilinear polygonal domains.

6.1.1 Hardness Results

Our hardness proofs are by modifying existing proofs of NP-hardness of the minmax versions of the *disjoint paths in planar graph* problem (MDPP), which, in general is stated as follows.

Given A planar graph $G = (V, E)$, vertices $s, t \in V$, a bound $B \in \mathbb{Z}^+$, a length function $l : E \mapsto \mathbb{Z}^+$

Find K (internally) vertex-disjoint paths connecting s and t, such that the length of each path is not more than B

The idea is that vertex-disjoint paths in G correspond to thick non-crossing paths in polygonal domain, created by "fattening" the edges of G drawing. The only challenge in transforming the graph problem into the geometric one is laying out the graph in the plane so that the lengths of the edges of the graph equal to the lengths of their images. After this is done (and the drawing is scaled, if necessary), each polygonal path l, representing an edge in G, is replaced by $(l)^1$. This way, disjoint paths in G correspond to non-crossing thick paths in the domain.

Strong NP-Hardness

The following definition is taken from [107].

Definition 6.1. *[107] The pairs of terminals (and the paths between them) in $\mathcal{ST}$ are* in parallel *if $\{s_1, \ldots, s_K, t_K, \ldots, t_1\}$ appear in this order around bdP^1.*

If the pairs of terminals are in parallel, the tree of slices $\mathcal{T}_{\mathcal{ST}}$ is a path.

Theorem 6.2. *The minmax version of the thick non-crossing paths problem in rectilinear domain is strongly NP-hard both in the L_1 and the Euclidean metrics, even if the paths are in parallel.*

Proof. In [72] Holst and Pina proved that MDPP is NP-complete even if $l : E \mapsto \{1, 2, 3\}$. Figure 1 in [72] (which we reproduce here in Fig. 14) shows the graph that was used to prove the hardness. The graph can be easily transformed into a polygonal domain, and a placement of terminals in the domain can be defined, such that the graph problem is feasible if and only if there exist length bounded disjoint thick paths between the pairs of the terminals.

First of all, the edges of G can be drawn in the plane so that the length of each edge is its Euclidean (or L_1) length: the length 1 edges will remain unchanged, the length 2 and 3 edges will "wiggle" a bit to gain length. The edges adjacent to s and r will be dropped; the nodes adjacent to s and r will become the terminals. Now, the drawing of the graph can be scaled so that after each straight line segment l, representing an edge in the graph is replaced by $(l)^1$, edge-disjoint paths in the graph still correspond to non-crossing thick paths in the domain.

Observe that the terminals pairs are in parallel. $\square$

Weak NP-Hardness for $K = 2$

If two thick paths have the same start s and destination t, we extend the notion of being non-crossing requiring that the paths diverge immediately after leaving s and meet again only at t:

Definition 6.3. *Let s, t be two points in a rectilinear domain. Two thick s-t paths Π_1 and Π_2 are* non-crossing *if $\Pi_1 \cap \Pi_2 = (s)^1 \cup (t)^1$.*

Theorem 6.4. *The minmax version of the two thick non-crossing rectilinear paths in rectilinear domain is weakly NP-hard both in the L_1 and in the Euclidean metrics, even if the paths are monotone.*

Proof. We start our reduction from the Two Short Disjoint Paths in a Weighted Planar Graph Problem:

Given A planar graph $G = (V, E)$, vertices $s, t \in V$, a bound B, a length function $l : E \to \mathbb{Z}^+$

Find Two (internally) vertex-disjoint paths connecting s and t such that the length of each path is not more than B

Holst and Pina [72] proved the problem is weakly NP-complete by a reduction from Partition: given m integers $c_1, \dots, c_m$, find a set $I \subseteq M = \{1, \dots, m\}$ such that $\sum_{i \in I} c_i = \sum_{i \in M \setminus I} c_i$. Indeed, the graph G_0 in the Figure 15 has two internally vertex-disjoint paths from s to t each of length bounded by $B_0 = 2(m + 1) + 1/2 \sum_{i=1}^{m} c_i$ if and only if the corresponding instance of Partition is feasible.

Starting from G_0, we build another graph, G (Fig. 16), such that G contains two vertex disjoint s-t paths each of length bounded by $B = 10m - 6 + 2 \sum_{i=1}^{m} c_i$

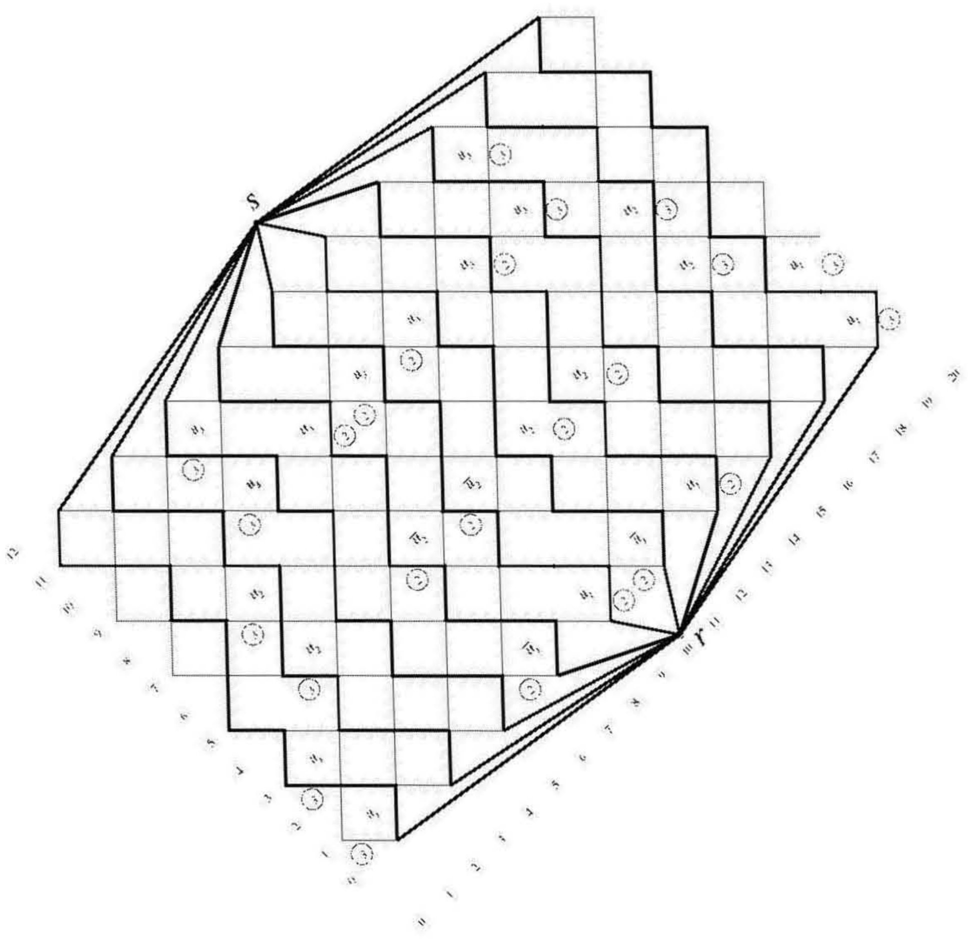


Figure 14: Figure 1 from [72], rotated by 45 degrees to represent a rectilinear domain. Circled numbers show the edge weights; letters show the regions corresponding to certain variables in the 3SAT instance.

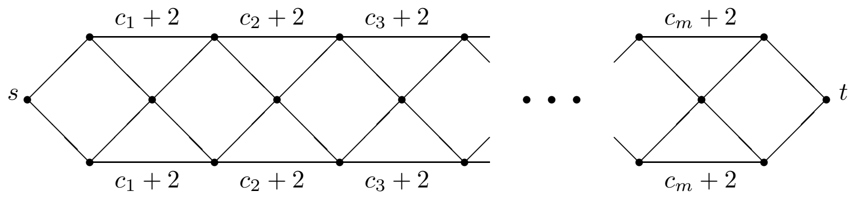


Figure 15: G_0. The edges with no label close to them are unit-length edges.

if and only if G_0 contains two vertex disjoint s-t paths each of length bounded by B_0.

Let $\mathcal{G}$ be the planar layout of G as in Figure 16. ($\mathcal{G}$ is the set of line segments of corresponding lengths representing the edges of G; the line segments meet at the points representing the vertices of G.)

Let P be the Minkowski sum of $\mathcal{G}$ and the unit square (Fig 17). By the construction of P, it contains two thick non-crossing s-t paths each of length bounded by B if and only if G contains two vertex disjoint s-t paths each of length bounded by B. Moreover, each of the paths in a solution to the problem is monotone. □

Corollary 6.5. *Unless P=NP, there exists no Fullypolynomial-Time Approximation Scheme for the problem of finding two thick non-crossing (monotone) (rectilinear) paths with bounded length in rectilinear domain.*

Proof. If there does not exist two paths of length at most B each, then the longest of the paths has the length at least $B+1$. Suppose, that there exists an algorithm that, for any $\epsilon > 0$, finds a solution, in which the longest path is at most $1+\epsilon$ times longer than the optimal; and that such an algorithm runs in time, polynomial in $1/\epsilon$. Take $\epsilon < \frac{1}{B+1}$. Then, the algorithm would output a solution of cost less than $B+1$ if and only if the corresponding instance of Partition is feasible. □

6.1.2 Pseudopolynomial-Time Algorithm

In the previous subsection we showed that finding two thick non-crossing rectilinear paths of bounded length in rectilinear domain is weakly NP-complete. In this subsection we show that if s_1, t_1, s_2, t_2 belong to the same obstacle, the problem can be solved in pseudo-polynomial time by a simple reduction to a recently solved problem in graph theory.

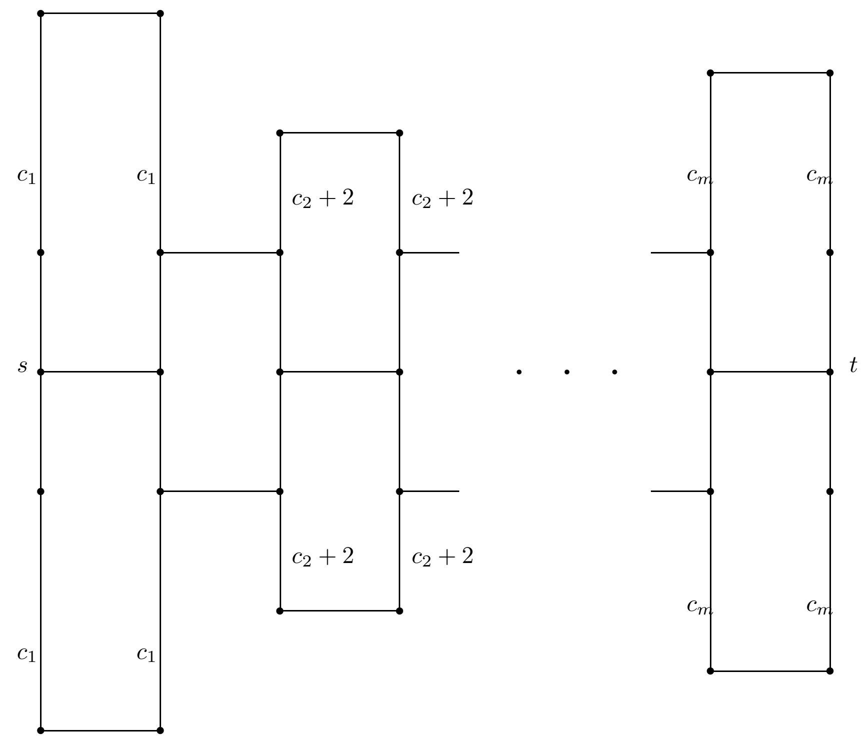


Figure 16: G. The edges with no label close to them are length-2 edges.

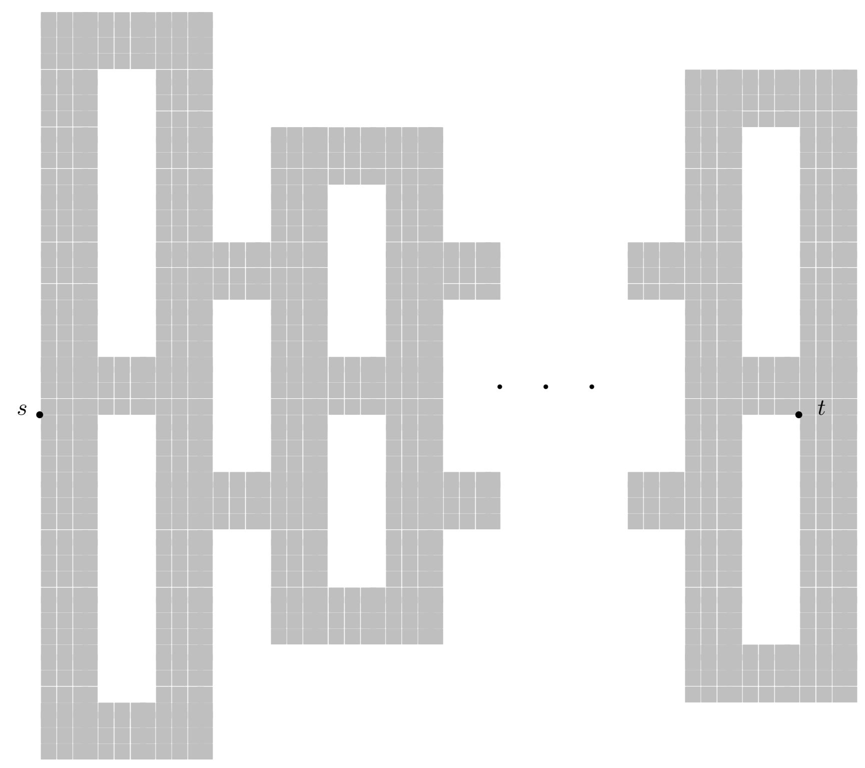


Figure 17: P. $\mathcal{G}$ is not drawn for clarity.

Assume that the coordinates of the obstacles are given with integers and that the whole domain resides in an N-by-N bounding square. Lay down an N-by-N integer grid so that the vertices of the obstacles snap onto the grid. Delete from the grid the nodes that are strictly inside the obstacles; no path could possibly go through the deleted nodes.

The obstacles edges can be classified into 4 types: North, East, South and West — in the intuitive way. We delete from the grid the nodes that belong to a South or a West edge; the reason is that a thick path with the reference path, visiting such a node, would intersect the interior of the obstacle. Let $\mathbb{G}$ be the grid graph induced by the remaining grid nodes; let s_1, t_1, s_2, t_2 be the vertices of $\mathbb{G}$ corresponding to the locations of s_1, t_1, s_2, t_2.

Lemma 6.6. *Let* $\Pi_1 = (\pi_1)^1$ *and* $\Pi_2 = (\pi_2)^1$ *be optimal thick non-crossing* s_1-t_1 *and* s_2-t_2 *paths. Then there exist thick non-crossing* s_1-t_1 *and* s_2-t_2 *paths* $\Pi_1^* = (\pi_1^*)^1$ *and* $\Pi_2^* = (\pi_2^*)^1$ *such that* π_1^* *and* π_2^* *are vertex-disjoint paths in* $\mathbb{G}$*, and* $|\pi_k^*| \leq |\pi_k|$*,* $k = 1, 2$*.*

Proof. The proof is by standard "snapping" argument, extensively used in rectilinear computational geometry to show path-preserving properties of finite graphs built from rectilinear domains, see, e.g., [127], survey [87] and references there off. The care must be taken though to ensure that both paths are snapped onto the grid without crossing.

Without loss of generality, suppose that there exists a *vertical* link not following the grid. Let l be the leftmost such link; say, the link belongs to π_1. Let x be the abscissa of l; let y_1 and y_2 be the ordinates of its endpoints (Fig. 18).

Proposition 6.7. *The open segment* $\mathcal{S}_1 = ((\lfloor x \rfloor, y_1), (\lfloor x \rfloor, y_2))$ *does not belong to* Π_1 *or* Π_2*.*

Proof. Otherwise, there exists a vertical edge of a reference path with abscissa strictly between $\lfloor x \rfloor - 1$ and $\lfloor x \rfloor$ and so l is not the leftmost vertical link running off the grid. □

Thus, we can push l to the left to snap it onto $\mathcal{S}_1$. If the two horizontal links adjacent to l are at different sides of the line supporting l, the length of π does not change with the snapping. If the two adjacent horizontal links both lie to the left of l, the path only shortens (and thus could not have been an optimal path). If the two segments both lie to the right of l, we will push l to the right (thus shortening the path).

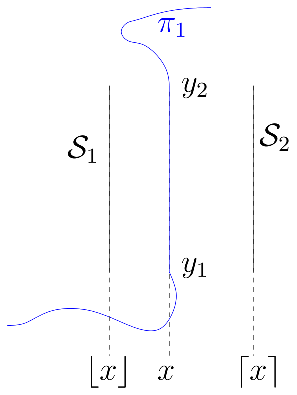


Figure 18: ℓ can be snapped either to the left or to the right without lengthening the path.

Proposition 6.8. *The open segment* $\mathcal{S}_2 = ((\lceil x \rceil, y_1), (\lceil x \rceil, y_2))$ *does not belong to any obstacle edge.*

Proof. Otherwise, $(l)^1$ would intersect the obstacle. □

Thus, the integer points in $\mathcal{S}_2$ belong to the grid and we can snap l onto it. If (parts of) Π_1 or Π_2 run through $(\mathcal{S}_2)^1$, by the same argument as above, the corresponding integer points at abscissa $\lceil x \rceil + 1$ are grid points and are free to have the path(s) snapped on them. If there are other parts of Π_1 and/or Π_2 that the new snapped path(s) intersect, those parts can be snapped to the right by the same argument.

The described snapping operation reduces the total number of the off-grid links in Π_1, Π_2 by at least 1, so after performing the operation finite number of times we obtain thick paths Π_1^*, Π_2^* as claimed in the Lemma. □

Theorem 6.9. *If* s_1, t_1, s_2, t_2 *belong to the same obstacle, the minmax version of the two thick non-crossing rectilinear paths problem in rectilinear domains can be solved in time, pseudopolynomial in the size of the input of the problem.*

Proof. By Lemma 6.6, the minmax version of the two thick non-crossing paths problem can be reduced to the minmax version of the two vertex-disjoint paths problem in the graph $\mathbb{G}$ built from the domain. It was shown in [72] that the latter problem can be solved in $O(m^4 L^2)$ time and $O(m^2 L^2)$ space for a planar graph with m nodes and maximum edge length L. The number of nodes in $\mathbb{G}$ is $O(N^2)$ and each edge length is 1. Since an instance of the thick non-crossing paths problem is specified with $O(n \log N)$ bits, the solution to the graph problem provides a pseudo-polynomial time algorithm for the geometric problem; the running time of the algorithm is $O(N^8)$. $\square$

We remark that in [72] it is also announced that a similar solution is possible for two more cases: that of routing 3 disjoint paths of bounded length, and that of arbitrary (but fixed K) when the paths are in parallel. Hence, our problem is solvable in pseudopolynomial time in the corresponding cases too.

6.1.3 Minimum-Cost Rectilinear Flows

We consider here the rectilinear version of the minimum-cost flow problem; we prove that the problem can be reduced to the flow problem on a path-preserving graph. It allows, under some restrictions, to solve the minsum version of the thick non-crossing paths problem.

The special case of the flow problem that we consider here can be defined as follows:

Given A collection of rectilinear obstacles and 2 pairs of points (without loss of generality — vertices of the obstacles): *sources* s_1, s_2 and *sinks* t_1, t_2. Obstacles coordinates are specified with integers.

Find Two thick non-crossing paths of minimum total length, one path connecting s_1 to one of $\{t_1, t_2\}$, and the second connecting s_2 to the other of $\{t_1, t_2\}$.

We call the described problem the *minimum-cost integer rectilinear flow* problem since we can think of the paths as of a flow from $(s_1)^1 \cup (s_2)^1$ to $(t_1)^1 \cup (t_2)^1$ with discrete thick flow lines represented by the paths in uniformly unit-capacitated free space. Although the problem bears close resemblance to the thick disjoint paths problem, there is an important difference: when routing the flows, we do not care about the source-destination pairings.

We solve the flow problem by reducing it to the flow problem in a graph $\mathbb{G}_0$ – a sparse version of the graph $\mathbb{G}$ from the previous subsection. We build $\mathbb{G}_0$ as follows. Take a North edge e of an obstacle in the domain. Extend the edge as far as possible both ways through the free space (until it collides with another obstacle or the bounding box). Shift the edge *up* by 1 and extend in the same way to run through the free space as far as possible. Do so for all North edges. Similarly, extend vertically each East edge and its shifted to the *right* copy. Take a South edge of an obstacle. Shift it *down* by 1 and extend horizontally. Also, shift it down by 2 and extend. Do so for all South edges. Similarly, extend vertically each West edge shifted to the *left* by 1 and by 2. The vertices in $\mathbb{G}_0$ will be the points of intersection of the extended segments;

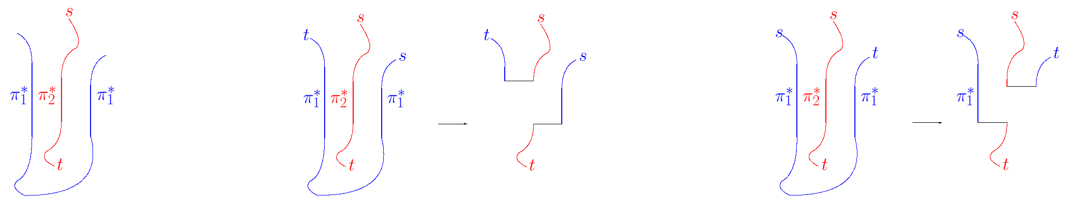


Figure 19: Left: Three paths pushed together. Center and right: The total length of the paths can be decreased by local modifications.

the edges will connect consecutive vertices. To complete the construction of $\mathbb{G}_0$, remove from it any edge b such that $(b)^1$ intersects an obstacle; add super-source s connected with 0-cost edges to the sources and super-sink t connected with 0-cost edges to the sinks. Let the capacity of each *node* in $\mathbb{G}_0$ be 1.

Any maximum (of value 2) s-t flow in $\mathbb{G}_0$ can be decomposed into two paths connecting the sources and the sinks in the domain. Indeed, by construction, the edges of $\mathbb{G}_0$ are at distance at least 1 from each other and from the obstacles (except, possibly, at the endpoints, where node capacity prohibits paths crossing). Thus the flow in $\mathbb{G}_0$ provides a feasible pair of thick paths in the domain. We show that the converse is also true:

Lemma 6.10. *Let Π_k, $k = 1, 2$ be optimal thick paths connecting sources and sinks, with reference paths π_k. Then there exist optimal thick s_k-t_k paths Π_k^* with reference paths π_k^* such that π_k^* are vertex-disjoint paths in $\mathbb{G}_0$, and $|\pi_1^*| + |\pi_2^*| \leq |\pi_1| + |\pi_2|$.*

Proof. Consider the snapping operation that was the basis of Lemma 6.6 proof. We snapped an edge l of path π_1 to one of the nearest grid lines. Instead, we could have pushed l until it collides with an obstacle or with another path segment; in the latter case we could push the paths together until the extreme path in the pushed group collides with an obstacle. What we need to show is that there will never be more than 2 paths in such a group.

First of all, if l collides with another edge c of the same path π_1, the path could be shortened by short-cutting from l to c. Thus, we can assume that the subpaths in the pushed group are alternating — π_1, than π_2, than π_1 again and, possibly, so on; assume that there are at least 3 paths in the group. Then, depending on the location of the sources and sinks, the total length of the paths can be decreased by performing local modifications (Fig. 19). $\square$

We remark that the source-sink pairing changes after the modification is applied, but in the problem we consider it is allowed.

Theorem 6.11. *The minimum-cost integer rectilinear flow problem can be solved in $O(n^4 \log n)$ time and $O(n^2)$ space.*

Proof. Construction of $\mathbb{G}_0$ can be accomplished in $O(n^2 \log n)$ time and $O(n^2)$ space. Indeed, in total $O(n \log n)$ time we can determine what the extended obstacles edges are [55, 114]. The intersections between the $O(n)$ extended segments can be computed in $O(n^2)$ time by brute force. For each of the $O(n^2)$ edges of $\mathbb{G}_0$ one can determine in $O(\log n)$ time whether the Minkowski sum of the edge and the unit square intersects an obstacle. Anyway, the time of the construction is dominated by finding the minimum-cost flow in $\mathbb{G}_0$, which can be done in $O(n^4 \log n)$ [104] since $\mathbb{G}_0$ is planar. $\square$

Under certain restrictions on the placement of s_1, s_2, t_1, t_2, the problem of minimizing the total length of s_1-t_1 and s_2-t_2 thick paths can be naturally reduced to the flow problem. For example, it is so if s_1, s_2, t_2, t_1 belong to the same obstacle and appear on the boundary of the obstacle in that order. Thus, in this case, the *minsum* version of the two thick non-crossing paths admits a polynomial time solution; this supports the observation that, like in graph theory, the minmax version is harder than the minsum.

The proposed algorithm naturally extends to the case of arbitrary number $2K$ of sources and sinks. The K paths, optimal for the minsum version of the problem, provide a K-approximation for the minmax version. Thus,

Corollary 6.12. *The minimum cost integer rectilinear flow can be found in $\tilde{O}(K^4 n^4)$ time and $O(K^2 n^2)$ space. If $s_1, \ldots, s_K, t_K \ldots, t_1$ belong to the same obstacle and appear on the boundary of the obstacle in that order, the optimal minsum short thick non-crossing paths (and a K-approximation to the minmax optimal paths) can be found within the same bounds.*

Our algorithms can be extended to higher dimensions.

6.2 Thin Paths in Polygons with Holes

As we mentioned earlier, finding short non-crossing paths between pairs of terminals is NP-hard even if the paths are thin [15]. In order to consider a tractable problem, we restrict our attention to the following formulation. Let P be a polygon, let $\mathcal{H}$ be the set of holes in it. We assume that every terminal in $\mathcal{ST}$ resides either on the boundary of P or on the boundary of one particular hole $H \in \mathcal{H}$. Without loss of generality, we assume that no pair of terminals has both its start and its destination on bd$H$, i.e., we assume that $s_1, \dots, s_K \in \text{bd}P$, and that the Positioning assumption still holds.

In [107], the problem of routing paths of minimum *total* length (the minsum version, in our terminology) in a polygon with 1 hole, $|\mathcal{H}| = 1$, was investigated; the minmax version of the problem was left unexplored. It is interesting to see how different the minsum and the minmax optimal solutions may be and, in particular, which paths may be expected to be as short as possible given the existence of the other paths. It is also interesting to investigate this question for polygons with more than one hole, $|\mathcal{H}| > 1$.

Partition $\mathcal{ST}$ into $\mathcal{ST}_P = \{(s_i, t_i) \,|\, s_i, t_i \in \text{bd}P\}$ — the set of pairs with both terminals on bd$P$ and $\mathcal{ST}_H = \{(s_k, t_k) \,|\, s_k \in \text{bd}P, t_k \in \text{bd}H\}$. The problem naturally splits into two cases:

Case I: $\mathcal{ST}_H = \emptyset$. It was observed in [107] that in this case the shortest paths are all-shortest. Considering the two possible routings of a path around H (H above the path or below the path) and comparing the solutions, the total time to route all paths is made linear in [107]:

Lemma 6.13. *[107] If $\mathcal{ST}_H = \emptyset$, all-shortest paths can be found in linear time.*

Case II: $\mathcal{ST}_H \neq \emptyset$. As observed in [107], after the homotopy type of an s_k-t_k path in $\mathcal{ST}_H$ has been chosen (clockwise around H or counterclockwise), the other paths are routed in a unique way. Thus, in linear time one could route the s_k-t_k path both clockwise and counterclockwise, and compare the total length of the paths under the different routings of s_k-t_k path. Thus,

Lemma 6.14. *[107] If $\mathcal{ST}_H \neq \emptyset$, the minsum version of the problem can be solved in linear time.*

Similarly, of course, the minmax version can be solved. Thus,

Corollary 6.15. *If $\mathcal{ST}_H \neq \emptyset$, the minmax version of the problem can be solved in linear time.*

The results in the lemmas can be combined in two corollaries:

Corollary 6.16. *[107] In a simple polygon with one hole, the shortest paths connecting the pairs without a terminal on the boundary of the hole are all-shortest.*

Corollary 6.17. *[107] In a simple polygon with one hole, both the minsum and the minmax versions of the shortest paths problem can be solved in linear time.*

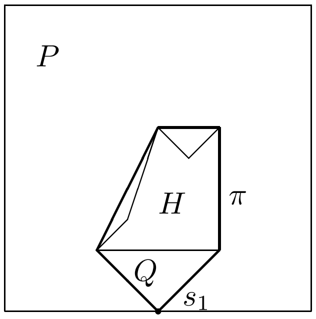


Figure 20: $\mathcal{ST}_P \cap Q = \emptyset$. Cf. Fig. 11 in [107].

We observe that Corollary 6.16 is also true even if the number of the holes is greater than one.

Lemma 6.18. *The paths connecting the terminals in $\mathcal{ST}_P$ are all-shortest.*

Proof. **Case I: $\mathcal{ST}_\mathcal{H} = \emptyset$.** In this case we prove the lemma by induction on K. Obviously, if $K = 1$, the lemma holds. If $K > 1$, let $\pi_1 \dots \pi_{K-1}$ be the collection of the shortest s_k-t_k paths, $k = 1 \dots K-1$; by the inductive hypothesis all of them are shortest. Let π_K be the shortest s_K-t_K path. Suppose, π_K intersects a path π_j, $1 \leq j < K$. By the Positioning assumption, s_j and t_j are either both below π_K or both above π_K. In any case, π_j intersects π_K an even number of times, and one of the paths could be strictly shortened by following the other path between consecutive points of intersection.

Case II: $\mathcal{ST}_\mathcal{H} \neq \emptyset$. Let $(s_1, t_1) \in \mathcal{ST}_\mathcal{H}$. Let π be the shortest path, going around H from s_1 to s_1 (Fig. 20). Let Q be the open subset of P strictly inside π. Clearly, $\mathcal{ST}_P \cap Q = \emptyset$, since $\mathcal{ST}_P \subset \text{bd}P$ and $Q \cap \text{bd}P = \emptyset$. Thus, if a shortest s_i-t_i path, $(s_i, t_i) \in \mathcal{ST}_P$, intersects π it must intersect it an even number of times. But since π is the shortest ("pull taught") path, the subpath of π_i between consecutive points of intersection with π could have been shortened by following π, a contradiction to the optimality of π_i. $\square$

Remarks

1. It is not true that each path in the optimum is the shortest path between its terminals (Fig. 21); the definition of all-shortest paths only requires that each path is as short as possible *given* the existence of the other paths.

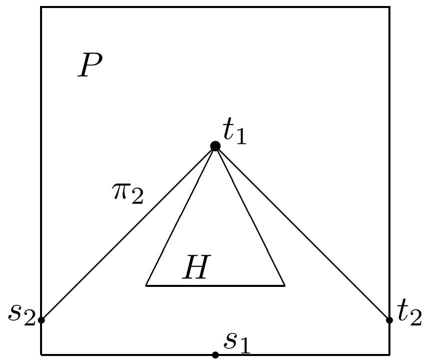


Figure 21: π_2 is not the shortest s_2-t_2 path, but it is the shortest possible given the existence of an s_1-t_1 path.

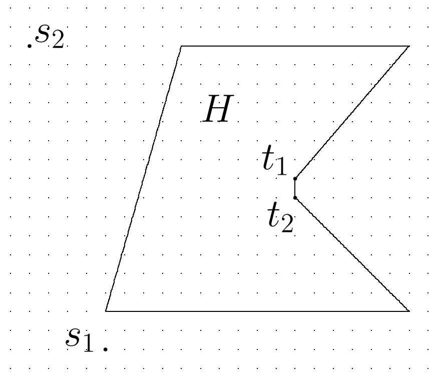


Figure 22: The minsum optimal paths go clockwise around H; the minmax optimal paths go counterclockwise.

2. The example in Fig. 22 shows that the paths in $\mathcal{ST}_{\mathcal{H}}$ optimal in the minsum and the minmax versions can be different.

3. It remains an open problem to find an efficient algorithm for actually routing the shortest paths in polygons with holes. In principle, nothing should change from the case of simple polygons: after some paths have been routed, a subsequent path could use the routed paths to progress faster towards the destination. Unfortunately, the funnel paradigm does not apply to the problem of finding shortest paths in polygons with holes. The existing approaches — searching the visibility graph and continuous Dijkstra paradigm (see surveys [98, 97]) — do not allow to charge the work done when routing a path to the part of the boundaries

explored. The brute force solution — routing the paths one by one — runs in $O(Kn \log n)$ time. Constructing a data structure for two-point queries and then using it to find the paths between pairs in $\mathcal{ST}$ seems to be an overkill since the set of points to be queried is restricted to $\mathcal{ST}$. Nevertheless, as observed in [69], when the homotopy of the paths is given, the funnel paradigm applies; we used this fact for routing thick paths in polygons with holes (Section 4).

4. Of course, after the shortest paths are constructed, a data structure of linear size can be build such that a shortest path can be reported in time proportional to its complexity. The data structure is identical to the one used for the shortest paths in simple polygons (see [107] or Section 3.3 of this thesis).

6.3 Routing Maximum Number of Thick Paths

In this and the following sections we solve the extensions of the *continuous maximum flow problem* [95]. To agree with [95], we slightly redefine our notation, and introduce some new notions.

For a set $Q \subset \mathbb{R}^2$ let bd$Q$ be its boundary. Whenever we speak of a set Q that has interior points ($Q \neq \text{bd}Q$), we will assume that Q is open, i.e, $\forall p \in \text{bd}Q, p \notin Q$. Let Ω be a polygonal domain defined by the outer (simple) polygon P and a set $\mathbb{H}$ of h holes, $H_1 \dots H_h$, in it. We will assume that no path can run outside Ω. Let n denote the number of vertices on the boundary of Ω.

Let $\Gamma_s, \Gamma_t \in \text{bd}P$ be two edges of P—the *source* and the *sink*. Then the set $\text{bd}P \setminus (\Gamma_s \cup \Gamma_t)$ consists of two connected components, T ("top") and B ("bottom"); Γ_s, B, Γ_t, T appear in this order counterclockwise on bd$P$. Since T and B cannot be penetrated by paths and flows in Ω, we will add them as holes $H_0 = T$ and $H_{h+1} = B$ to $\mathbb{H}$. Abusing notation, we will assume that $\Gamma_s, \Gamma_t \notin \text{bd}\Omega$, i.e., that $\text{bd}\Omega = \text{bd}\mathbb{H}$.

We will assume that Ω has already been "perforated" at Γ_s, Γ_t, and Riemann flaps were glued to Ω at the perforated edges [95]. As described in [95], this allows one to avoid complications that may arise when a thick path, a wavefront, a Voronoi edge, etc. "protrudes" through Γ_s or Γ_t.

A (thin) *path* is a simple curve, within Ω, whose one endpoint belongs to Γ_s, the other – to Γ_t. Note that we only consider source-sink paths. By a *thick path* we will actually mean the canonical part of it. Thus, any path starts/ends normally to the source/sink. By our results from Section 5, when

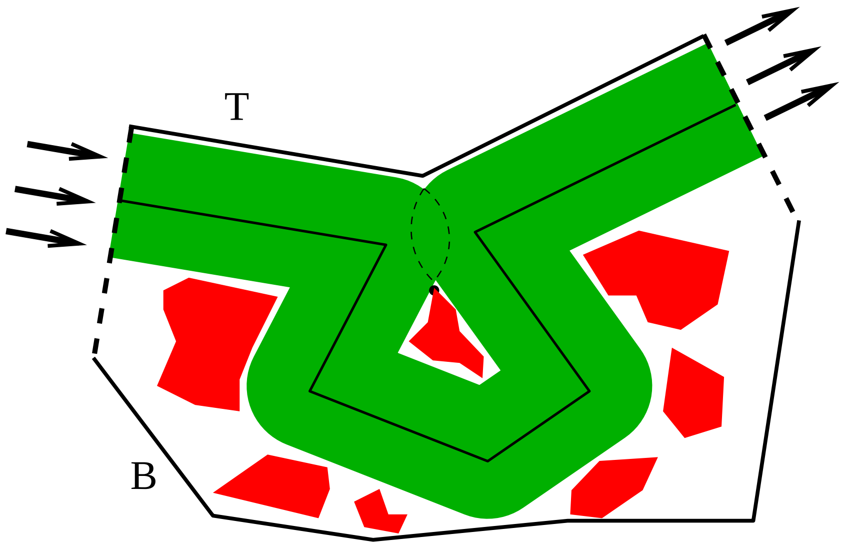


Figure 23: Only self-overlapping thick path exists.

working with the canonical parts of thick paths, this does not imply any loss of generality.

In this section we consider finding the maximum number of thick non-crossing paths in Ω. We modify the continuous-Dijsktra-type uppermost shortest path algorithm for finding the maxflow [95], to take into account the discrete nature of our problem.

Thick Paths May Self-Overlap By definition, the reference path of a thick path is simple, i.e., non-self-intersecting. At the same time, the definition allows a thick path to be self-overlapping. In fact, in some instances only self-overlapping thick paths exist (Fig. 23, see also Fig. 4 in [76]). This means, in particular, that routing a thick path is different from routing a thick *wire*—a non-self-overlapping thick path. While a thick path can be found by the standard procedure of offsetting the obstacles and then searching for a thin path within the modified free space, we are not aware of *any* algorithm for finding a thick wire. To the best of our knowledge, this difference between paths and wires has not been noticed anywhere but in [76]. This may be due to the fact that in wire-routing applications several special cases were generally considered—monotone paths, routing in simple polygons, rectilinear domains. In these cases, paths and wires are the same.

Having acknowledged the possibility of the self-overlap, we show how to find

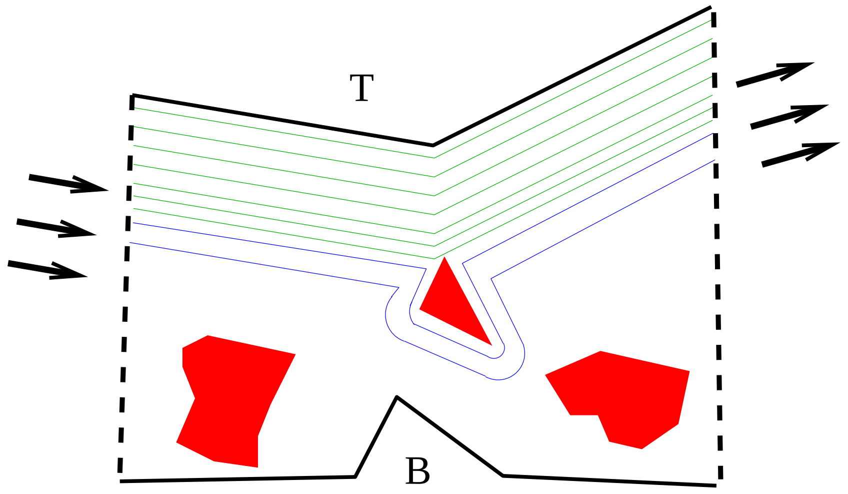


Figure 24: The wavefronts make up the streamlines of the flow. After hitting an obstacle, the streamlines start going around it.

the maximum number of thick paths using a modification of the continuous uppermost path algorithm [95]. The modification involves attaching, where necessary, Riemann flaps to allow for a self-overlapping thick path to remain non-self-overlapping in the modified domain.

The algorithm of [95] uses the following "grass fire" analogy. Suppose that the free space $\Omega \setminus \mathbb{H}$ is grass over which fire travels at speed 1. Suppose also that the holes are composed of a highly flammable material (the fire travels through a hole at infinite speed) so that as soon as the fire hits a hole, its entire boundary immediately ignites. Ignite T at time 0. The *wavefront* at time τ is the boundary of the grass that has burnt by τ. The wavefronts make up the streamlines of the flow. The algorithm fills up the free space with the streamlines as the fire propagates until reaching B so that no more streamlines can be found. The *events* of the algorithm happen when the fire hits an obstacle. (There are also other events, not relevant here.) Two things happens at an event: the obstacle's boundary joins the wavefront, and the streamlines start going below the obstacle. Refer to Fig. 24.

We modify the grass fire approach as follows. Suppose that the fire has not hit a hole after burning for 2 units of time. Then we route a thick path inside the burnt grass. The CDFT ensures that a path exists. We argue below that routing a path as described can not hurt the existence of the other paths in a collection of the maximum number of thick paths. We then start over, treating the wavefront at 2 as new T. Refer to Fig. 25.

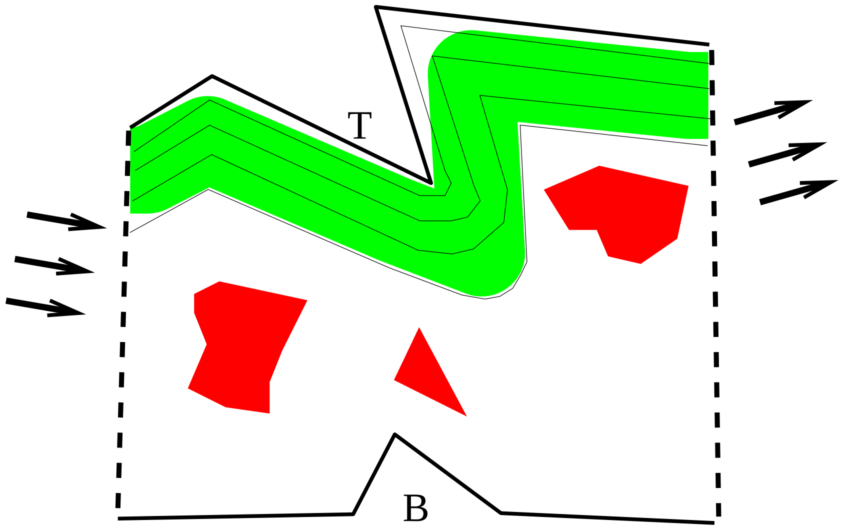


Figure 25: Event time is greater than 2. By the CFDT, a thick path can be routed within the grass that burnt for 2 units of time.

Suppose, on the other hand, that a hole H is hit by the fire at time $\tau < 2$. Let e be the segment of length τ that connects H to T; we assume that e is unique. We slit the free space along e and around bd$H$. This "carves out" H, and it is no longer a hole (since it is now *outside* the domain). We glue a Riemann sheet to each copy of e. In each sheet, we place a circular segment, of radius 2, having e as a chord. Refer to Fig. 26. Now we continue the grass fire. Not only we ignite the whole boundary of H, but also a belt of thickness τ around it. This belt represents the fact that the path that is being routed "jumps over" the hole and runs now on the other side of it. As before, when the grass has burnt for 2 units of time without hitting a hole, we route a thick path within the burnt grass. Carving out the holes and attaching the circular segments ensures that we are routing within a simple polygon; thus, the CFDT may be applied to show the existence of a thick path in it. We argue that when dropped to the base sheet $\mathbb{R}^2$ (where Ω lives) the reference path of the routed thick path does not intersect itself, and that the thick path does not intersect any holes. As before, it can be argued that the routed path does not interfere with the other thick-paths-to-be-routed. Also as before, we treat the wavefront as the new top, and continue the propagation.

Remark. Observe, that we cannot simply bridge H to the top. Indeed, although no thick path can have H below (since the distance from H to T is less

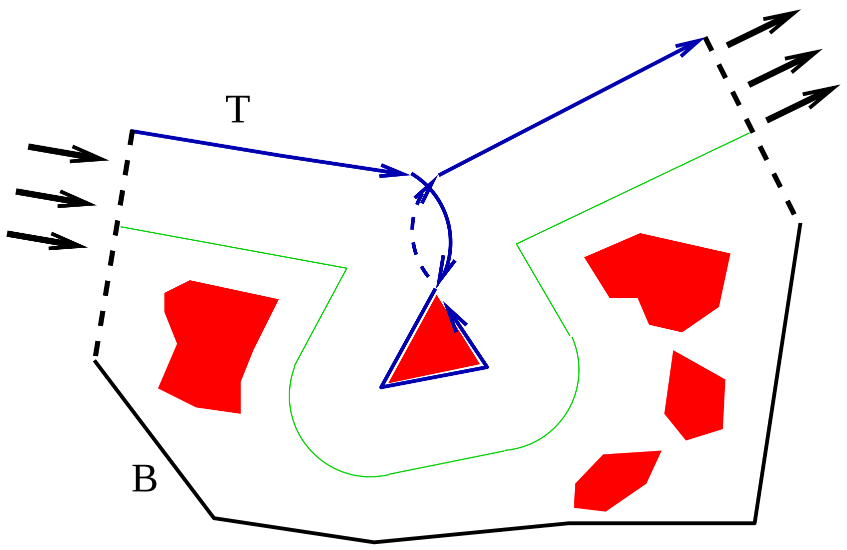


Figure 26: Event time is less than 2. Slit $\mathbb{R}^2$ along the shortest segment, connecting the hit hole to T, glue Riemann flaps along each copy of the slit edge, and place a circular segment inside each flap. The new domain excludes the hole. The arrows indicate clockwise traversal of the new top (blue). The new wavefront is shown green, its distance from the hole equals the event time.

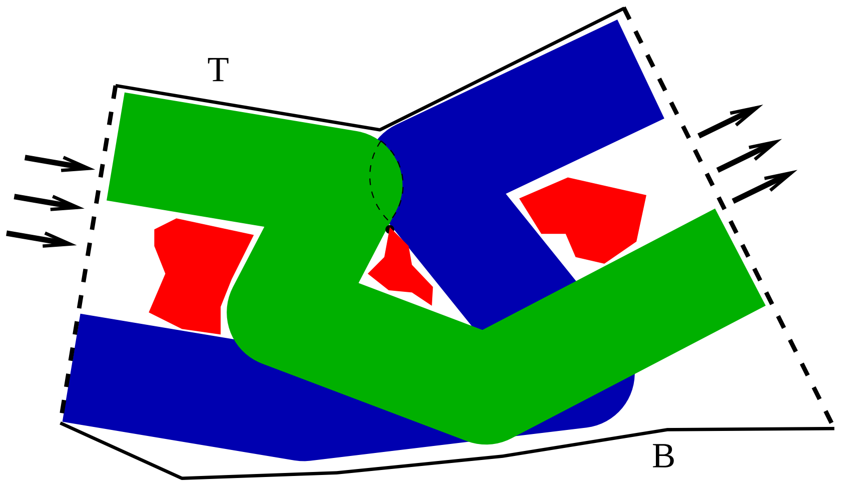


Figure 27: There is no unique bridge to T that would not be crossed by *any* thick path; different paths may "cut off" the hole from the top differently.

than 2) parts of a thick path may run "between" H and T. Moreover, different thick paths may occupy overlapping portions of the space between H and T (see Fig. 27). It shows that it is not possible to build a "bridge" between H and T, such that *no* thick path would ever cross the bridge. Even more, as the example in Fig 23 shows, just one thick path may completely "cut off" H from T so that although H is above the path, there is no path (bridge) in Ω between the hole and the top.

As described above, the running time of the algorithm is proportional to the output size, i.e., to the maximum number of thick paths in Ω; call this number K. We can remove this dependence by stopping the fire propagation *only* when a hole is hit. We give the details in the next theorem.

Theorem 6.19. *(A representation of) the maximum number of thick non-crossing paths can be found in* $O(nh + n \log n)$ *time.*

Proof. Let τ^* be the time of the first event, i.e., the time, when the fire reaches a hole, H, in Ω. Suppose that $\tau^* \geq 2$; let $W = \lfloor \tau^*/2 \rfloor \geq 1$. Let π be the wavefront at $2W \leq \tau^*$. The part of Ω between T and π is a simple polygon, Ω_π, in which a flow of value $2W$ exists; this follows from the maxflow algorithm of [95] (streamlines=wavefronts). By the CFDT (Theorem 5.5), there exists a W-thick path Π in Ω_π. By the Gluing Lemma (Lemma 5.2), Π is a representation of a set of W thick paths. We continue the propagation treating π as the new top.

Suppose now that $\tau^* < 2$. Since the distance from H to T is less than 2, the reference path of a thick path cannot intersect e (the shortest segment between H and T). Thus, the reference path of a thick path, which will be routed after the fire burns eventless for more than 2 units of time, does not intersect itself even if parts of the thick path intersect e. The parts of a thick path, that could possibly intersect e, belong to the circular segments, of radius 2, that have e as a chord (Fig. 26). The parts cannot intersect an obstacle as no obstacle intersects any of the circular segments. Indeed, the diameter of each of the segments is $\tau^* < 2$, thus, if a hole intersected a segment, the intersection point (and not the endpoint of e) would have been the event point. Overall, it shows that when the even if the thick path "bulges into" the circular segments, when dropped (projected) onto the base sheet $\mathbb{R}^2$, it remains a valid thick path.

To prove correctness of the algorithm we need to show that the W (uppermost) thick paths, produced by the algorithm after an event with $\tau^* \geq 2$, do not "block" routing (in the future) the remaining $K - W$ thick paths. This follows from the following fact: Let $(\Pi_1, \ldots, \Pi_K)$ be a set of K thick non-crossing paths in Ω; then the path Π_{W+1} does not intersect Ω_π. Indeed, the distance from π to T is $2W$, thus, if π were intersected by (the upper boundary of) Π_{W+1}, the W thick paths could not have "fit in" between Π_{W+1} and T.

As for the running time, the grass fire can be simulated in $O(nh + n \log n)$ time as described in [95]. There is $O(h)$ events. If $\tau^* < 2$, the modifications to the domain take constant time per event. Otherwise, a W-thick paths can be routed (in a simple polygon) in linear time (see, e.g., Theorem 3.10).

The algorithm outputs $K^* \leq K$ thick paths, where K^* is the number of different threadings of the paths through Ω. Using the thick paths, output by the algorithm, we can find, for any $k \in \{1 \ldots K\}$, the kth path Π_k in the collection of K thick paths in time, proportional to the combinatorial complexity of Π_k (Section 3.3). In particular, all K paths may be output in $O(Kn)$ time. $\square$

Remark. The running time of our algorithm matches that of the algorithm for the maxflow [95]. The bottleneck in both algorithms is the grass fire propagation in the "0/1 metric" [61], in which one travels at unit speed in the free space and at the infinite speed across the holes. It is possible that the $O(n \log n)$-time continuous Dijkstra algorithm of Hershberger and Suri [70] for shortest pats in polygonal domains may be extended to the 0/1 metric.

Discrete Continuous MaxFlow-MinCut Theorem The notion of the *critical graph* of the domain [61, 95] is central to finding shortest paths in 0/1 metric, and to mincuts and maxflows in geometric domains. The critical graph has a vertex for every hole in $\mathbb{H}$; the length of an edge (i, j) is the distance between the holes H_i and H_j in the 0/1 metric (essentially, it is enough to connect be edges only those holes, the shortest segment between which does not intersect other holes). Mitchell [95] used the critical graph to formulate and prove the "Continuous MaxFlow-MinCut Theorem" [78, 95, 117] —the continuous version of the famous network flow theorem:

Theorem 6.20. *[95] The value of the maximum flow in the domain equals to the length of the shortest T-B path in the critical graph.*

The results in this section establish the (oxymoronic) *Discrete Continuous MaxFlow-MinCut Theorem.* Here, as in [95] and above, "Continuous" refers to the continuous flow in a geometric domain, as opposed to a flow in a (discrete) network. The "Discrete" refers to the fact that the flowlines are not allowed to be arbitrarily thin: the (support of the) flow must decompose into a *finite* set of thick paths. To state our theorem, we introduce the *thresholded critical graph* $G_{\lfloor\rfloor}$: it is the critical graph, in which the length of every edge is divided by 2 and rounded down to the nearest integer.[10]

Theorem 6.21. Discrete Continuous MaxFlow-MinCut Theorem. *The maximum number of thick non-crossing paths in the domain equals to the length of the shortest T-B path in the thresholded critical graph.*

Proof. The events of our uppermost shortest path algorithm correspond to the wavefront reaching the holes. The number of the paths routed at each event equals to the length of an edge in the thresholded critical graph. By induction on the ordinal number of the event, the event time (since the ignition of T) equals to the shortest-path distance from T to the hit hole in $G_{\lfloor\rfloor}$. $\square$

The above theorem is the first Megner-type result for (thick) paths in continua.

6.3.1 Paths of Different Thicknesses

Consider the following decision problem: Given K numbers $w_1, \ldots, w_K$, does there exist a collection of $(w_1, \ldots, w_K)$-thick paths in Ω? If the order of the

[10]In [95] it was suggested to use the graph to find the maximum number of "well separated" paths; as we note in the next section, this approach does not always give a correct answer.

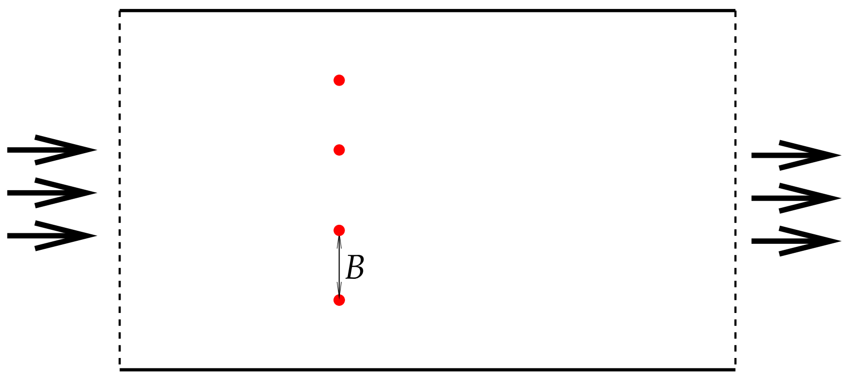


Figure 28: The 3-PARTITION problem asks if the numbers $a_1 \dots a_{3n}$, $\sum_i a_i = nB$, $B/4 < a_i < B/2$, can be split into n groups of 3, such that the sum of numbers in each group is B. The polygonal domain created from a 3-PARTITION instance has $n-1$ holes; each hole is a point. The distance between ith and $(i+1)$st hole is B. The paths' thicknesses are $a_1 \dots a_{3n}$; all paths can be packed into the domain if and only if the 3-PARTITION instance is solvable.

paths as they appear along Γ_s is given, the problem can be solved by the algorithm above. Otherwise, the problem is NP-hard by reduction from 3-PARTITION; see Fig. 28.

6.3.2 Well Separated Paths

The notion of non-crossing thick paths is distinct from that of "well separated" paths (and this may be a better model for ATM). The (thin) paths $\pi_1 \dots \pi_K$ are called *well separated* if the distance between any two of them is at least 2. As example in Figure 29 shows, the number of well separated paths can be arbitrarily larger than the number of thick paths that can be routed through a domain.

We can use the uppermost path algorithm to find the maximum number of well separated paths in a domain as follows. Let the first path follow T. Offset the path (i.e., T) by 2. Route the next path along the boundary of the offset; if a hole is encountered, follow the hole's boundary, leaving the hole above the path. Refer to Fig. 30. Repeat until B is reached by the offset path. The correctness of the algorithm follows from the argument, similar to the one in Theorem 6.19.

Remark. The paths, found by the above algorithm may not be (subjectively)

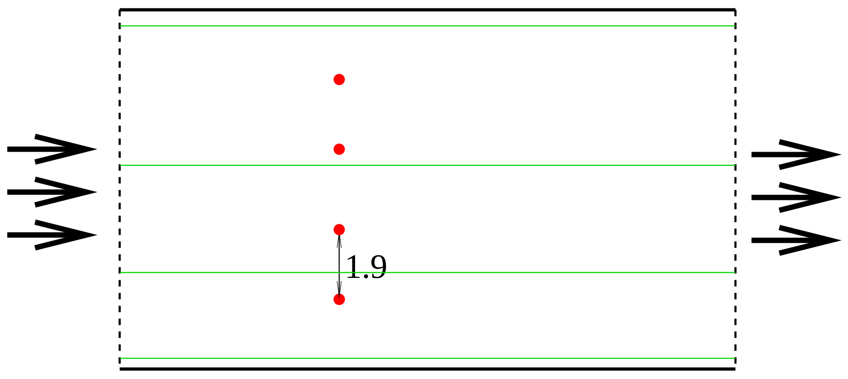


Figure 29: The distance between the (point) obstacles is 1.9. A thin path can be routed between almost every pair of consecutive obstacles; the paths will be well separated. No thick path can be routed.

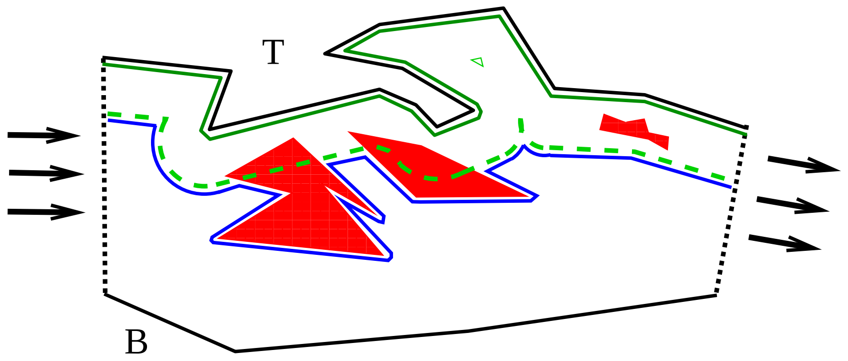


Figure 30: The first path (solid green) follows T. The offset of the path (dashed green) is the locus of points within Euclidean distance 2 from the path. The second path (solid blue) follows the offset and the boundaries of the holes that are "stuck" in the offset.

appealing to fly by (as, e.g., the paths in Fig. 30). It would be interesting to consider the problem of finding "flyable" well separated paths. Note that our thick paths look "nice", at least their curvature is never greater than 1; this is due to the paths being locally optimal (Lemma 3.11).

6.4 Simple Algorithms for Simple Polygons

In a simple polygon P ($\mathbb{H} = \{B, T\}$) faster algorithms are possible.

6.4.1 Maximum Number of Thick Paths

Let V^* be the value of the maximum flow in P. Let $\sigma^*_{V^*}$ be the *minimum-cost* flow of value V^*. By the CFDT, $\sigma^*_{V^*}$ is a $V^*/2$-thick shortest path. By the gluing Lemma 5.2, the path can be decomposed into $\lfloor V^*/2 \rfloor$ thick paths and one $(V^* - 2\lfloor V^*/2 \rfloor)$-thick path. Thus, $K \geq \lfloor V^*/2 \rfloor$, where K is the maximum number of thick non-crossing paths that can be routed in P.

On the other hand, consider a set of K *shortest* thick non-crossing paths in P. By the CDFT, there exists a Γ_s-Γ_t flow of value $2K$ in P. Thus, $V^*/2 \geq K$. Since K is an integer, we have:

Lemma 6.22. $K = \lfloor V^*/2 \rfloor$.

Theorem 6.23. *(A representation of) K thick non-crossing paths in P can be found in linear time.*

Proof. In [95] it was shown that V^* can be found in linear time after the medial axis of P is built, which can also be done in linear time [30]. After V^* is found, we offset bd$P$ by $\lfloor V^* \rfloor/2$. The free space is a splinegon, in which a path π between Γ_s and Γ_t can be found in linear time. We can inflate π also in linear time; the inflated path serves as a representation for the K thick paths in P. □

6.4.2 Monotone Thick Paths and Flows

We call a flow *monotone* if each of its streamlines is monotone w.r.t. the x-axis. We call a thick path Π monotone if its reference path π is x-monotone, i.e., if every vertical line intersects π in at most one point. It is easy to see that a monotone thick path is a monotone simple polygon: every vertical line intersects the path in one contiguous (possibly, empty) vertical segment.

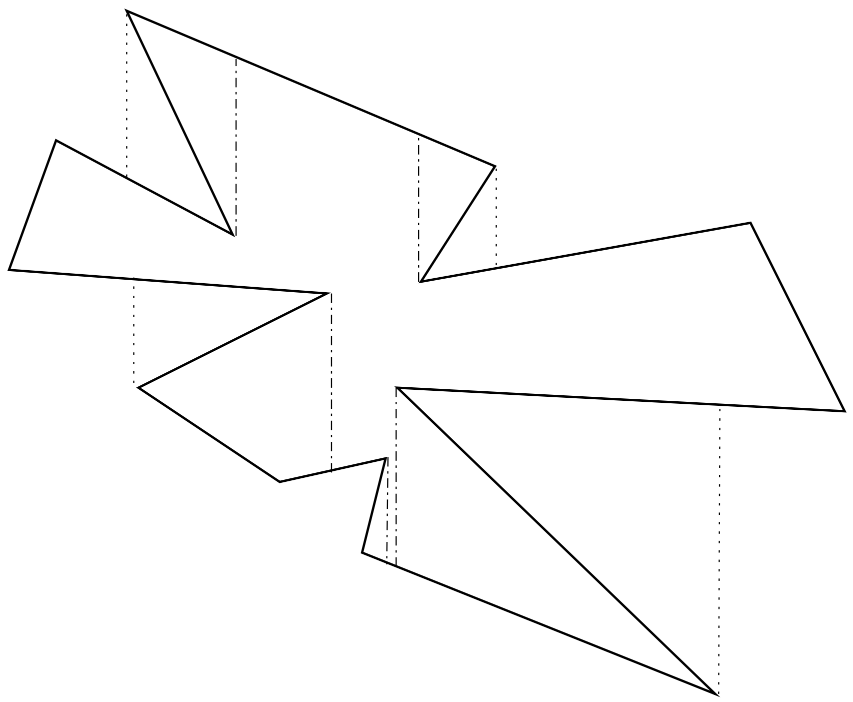


Figure 31: The waterfalls defining the inner (resp. outer) monotone hull are shown dashed-dotted (resp. dotted).

The algorithms from the previous subsection can be extended to find monotone paths and flows by "monotonizing" bd$P$. The *inner* (resp. *outer*) *monotone hull* of P is the largest (resp. smallest) monotone polygon that is contained (resp. contains) P. The inner hull can be obtained as follows. Sweep a vertical line in the x direction. For every vertex v of P, connect v to the first point of P hit when going up from v, and when going down. (Following [9], we call the connecting segments *waterfalls*.) This results in a trapezoidation of P. Starting from the leftmost trapezoid, attach to each trapezoid the trapezoid(s) to the right of it, until reaching the rightmost one. These trapezoids form the inner monotone hull, $M(P)$, of P, Fig. 31. The monotone hull of a simple polygonal chain on bd$P$ may be defined similarly, see Fig. 32.

Assume, for simplicity, that there is a unique trapezoid, containing Γ_s, and a unique trapezoid, containing Γ_t. Assume also, without loss of generality, that the trapezoid, containing Γ_s (resp. Γ_t), is the leftmost (resp. rightmost) one. By the CFDT, finding (minimum-cost) maximum flow is equivalent the routing a *thickest* path. The proof of the CFDT may be extended verba-

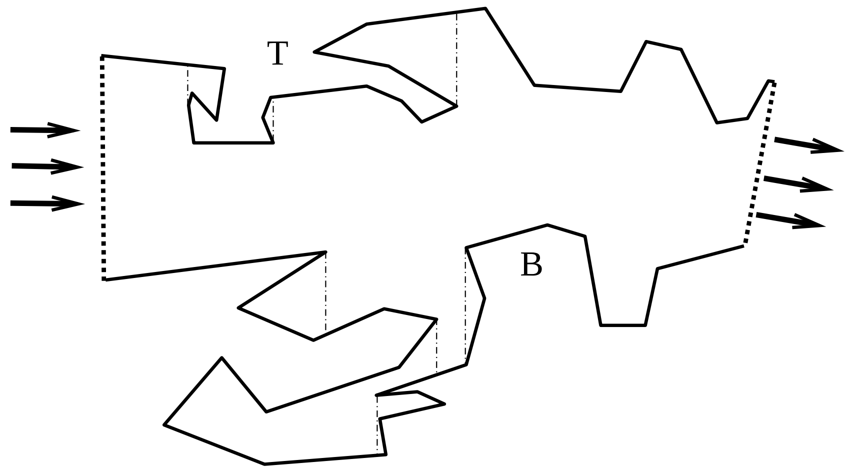


Figure 32: Waterfalls are dashed-doted. Monotonizing T and B makes P a monotone polygon; any shortest path in it is monotone.

tim to the monotone case: (the support of the) minimum-cost monotone flow is a thickest monotone path. Since a monotone path is a monotone simple polygon, it will not intersect $M(B)$ and $M(C)$—the monotonized B and T (see Fig. 32). Moreover, it is easy to see that any (locally) shortest path in the simple (monotone) polygon, defined by $\Gamma_s, M(B), \Gamma_t, M(T)$ is monotone. Thus, we can monotonize B and T, and solve the maximum flow problem in it—the flow will be the maximum monotone flow in the original polygon. By thresholding the flow (similarly to Section 6.4.1), one can obtain the maximum number of thick paths in the domain.

Theorem 6.24. *Finding the maximum monotone flow and the maximum number of thick paths in a simple polygon can be done in linear time.*

Proof. After triangulating the polygon, it can be monotonized in linear time. Then our algorithms from the previous subsection can be applied. $\square$

6.5 Monotone Paths and Flows in Polygonal Domains

In this section we extend the algorithms of Section 6.4.2 to finding maximum monotone flow in a polygonal domain with holes. We start by monotonizing T. We then run the uppermost shortest path maxflow algorithm [95], filling the free space with the flowlines. Since the top is monotone, the flowlines are also such. When a hole is hit by a wavefront, we outer-monotonize the hole. If

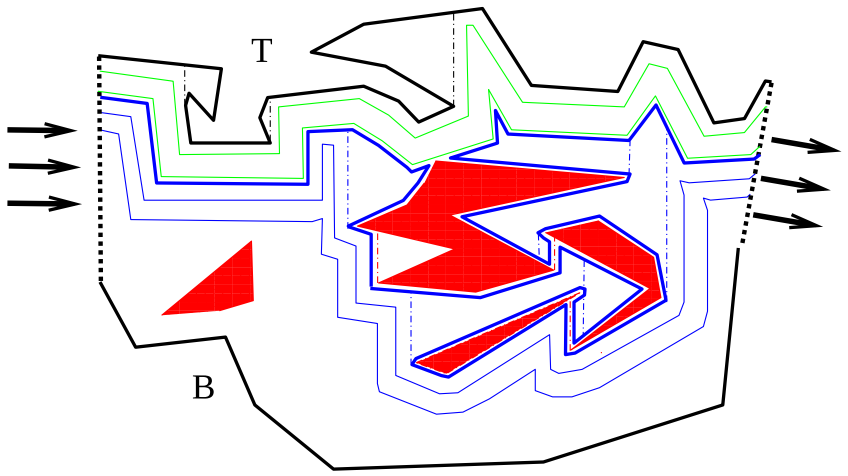


Figure 33: T is monotonized and the uppermost path algorithm is run until a hole is hit (green streamlines). The hole is monotonized (waterfalls are dashed-dotted red); if a waterfall hits a new hole, it is also monotonized. The wavefront and the monotonized holes' boundaries are the new top (thick blue). The new top is monotonized (waterfalls are dashed-dotted blue), and the and algorithm continues (blue streamlines).

during the monotonization a waterfall hits another hole, the hole is also (outer-) monotonized. The wavefront and the boundaries of the monotonized holes are assigned to be the new top. The new top is monotonized, and the grass fire continues (Fig. 33). See [9] for efficient algorithms for monotomization of the holes.

Theorem 6.25. *The maximum monotone flow and (a representation of) the maximum number of monotone thick non-crossing paths can be found in* $O(nh + n\log n)$ *time.*

Remark. As an alternative to monotonizing the holes "on-the-fly", we could have pre-monotonized all holes; if during the monotonization of a hole, another hole is hit by a waterfall, the holes are merged. After that, we could assume that all holes are monotone.

7 Related Problems

In this section we give solutions to several motion planning problems. In the *touring* problems the shortest tour that visits a given sequence of objects is sought. In our motivating applications, finding shortest paths *with bounded number of links* is an important extension of the shortest path problem. Finally, finding optimal tours through pixelated domains and grids is a classical problem in computational geometry and graph theory.

7.1 Touring Problems

Here we study the problem of finding a shortest tour visiting a given sequence of bodies. The difference between the problems studied here and classical TSP-like problems is that we assume that the sequence in which the bodies are to be visited is given in advance. An example of such touring problem is the problem of finding a shortest tour through a set of line segments [56, 99, 112, 115]. Due to the bit complexity of the solution, in L_2 one can only hope to solve the touring problem approximately in polynomial time. A singly-exponential time *exact* algorithm for the problem of touring line segments in $\mathbb{R}^3$ is given in [112]. In [24] the problem of finding ϵ-approximate shortest tour of *lines* in $\mathbb{R}^3$ is solved in time doubly logarithmic in $1/\epsilon$. The general problem for *convex polygons* in $\mathbb{R}^2$ under Euclidean metric is solved in [47].

7.1.1 L_1-Shortest Tours through a Sequence of Segments

We first consider touring line segments in the plane in L_1 metric. Given a sequence of K disjoint segments in the plane, a start point s and a target point t, we seek a path that starts at s, visits in order each of the segments, and ends at t, such that the L_1 length of the path is minimized. We give an $O(K^2)$ algorithm for the problem. As a by-product, our algorithm builds

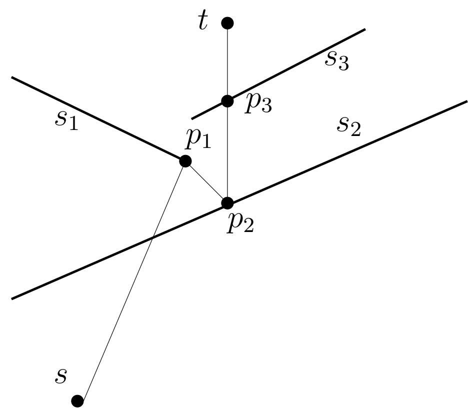


Figure 34: An example. p_1, p_2, p_3 are the first contact points. p_1 is a bend, p_2 is a reflection, p_3 is a pass-through.

a data structure, of size $O(K)$, such that the shortest path, visiting k first segments in the sequence, to any point in the plane can be output in $O(k)$ time.

Preliminaries

Let $s_1, \ldots, s_K$ be a sequence of K pairwise-disjoint segments in the plane; let s, t be two points. For $k = 1 \ldots K$ we say that a path π *visits* the sequence $s_1, \ldots, s_k$ if it starts at s, and there exist points $p_1^* \in s_1 \cap \pi, \ldots, p_k^* \in s_k \cap \pi$ such that $p_1^*, \ldots, p_k^*$ appear in order along π. For a point $z \in \mathbb{R}^2$ and $k \in \{1 \ldots K\}$, a *k-path* to z is a path to z that visits $s_1, \ldots, s_k$. Let $\pi_k(z)$ be an L_1-shortest k-path to z; let $|\pi_k(z)|$ denote its length. Our problem is then to find $\pi_K(t)$.

Let π be a path that visits $s_1, \ldots, s_K$. Let $p_0 = s$, and let p_k, $k = 1 \ldots K$, denote the first point of π (i.e., the point closest to s along the path) that lies on s_k and comes *after* p_{k-1} along π. We call the points $p_1, \ldots, p_K$ the *first contact points* (Fig. 34). We let $p_k(z) \subseteq s_k$ denote the set of the possible first contact points of $\pi_k(z)$ with s_k, i.e., the points p in s_k such that $|zp| + |\pi_{k-1}(p)|$ is minimum over all $p \in s_k$.

Without loss of generality we assume that no edge of the optimal tour goes through an input segment; we also assume, without loss of generality, that no segment is horizontal or vertical. We denote by R_k the smallest axis-aligned rectangle enclosing s_k. For a point $z \in R_k$ let $h_k(z)$, $v_k(z)$ denote the "horizontal" and "vertical" projection of z on s_k. We denote the endpoints of s_k by a_k, b_k (Fig. 35).

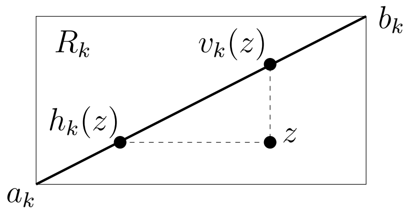


Figure 35: a_k, b_k, R_k, h_k and v_k.

As with the shortest Euclidean tours, we may consider, without loss of generality, only the tours that are polygonal chains that bend only at the segments. The first points of contact of the tour with the segments, $p_1, \ldots, p_K$, may be classified into three types: a *bend* at a vertex, a *reflection* off a point interior to the segment, and a *pass-through* (refer to Fig. 34).

The algorithm of [47], which solves the touring problem in L_2, builds a set of subdivisions $\mathcal{S}_k$, $k = 1 \ldots K$, such that for any point z in a cell of kth subdivision, $\pi_k(z)$ is in the same-type contact with s_k. The correctness of the algorithm is due to several facts:

1. A uniqueness lemma — in L_2 metric, for any $z \in \mathbb{R}^2$, and any $k = 1 \ldots K$, the path $\pi_k(z)$ is unique.

2. The vertices of the subdivision $\mathcal{S}_k$ are defined by the endpoints of the segment s_k.

3. The *reflection* of a point in a segment is a bona-fide notion.

None of the above holds in L_1 (Fig. 36). This complicates a straightforward extension the algorithm of [47] to L_1.[11] Nevertheless, using the ideas from [47], we build similar subdivisions for the L_1-shortest tours and solve the touring problem.

Our Contributions We prove that the subdivision $\mathcal{S}_k$, grouping the points in the plane, shortest k-paths to which are in the same-type contact with the segment s_k, has constant complexity for any $k = 1 \ldots K$. We show that the function $\ell_k(p), \ell_k : s_k \mapsto \mathbb{R}^+$ that gives the length of the L_1-shortest k-path to a point p on the segment s_k is a convex piecewise-linear function with

[11]The only problem, which we aware of, that is "harder" in L_1 than in L_2 is the problem of finding the distance between two convex polygons [125] (where the convexity is understood relative to the metric).

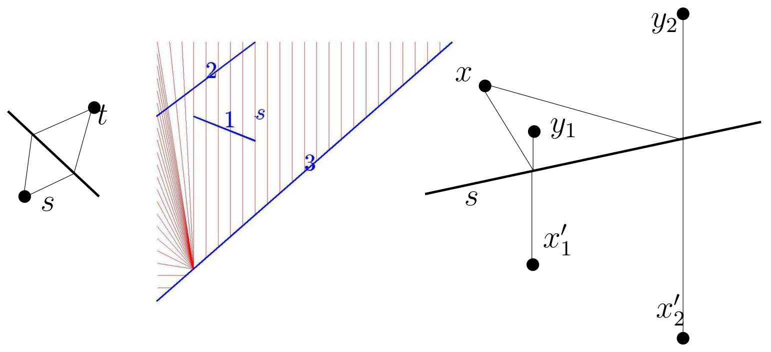


Figure 36: Left: L_1-shortest tours are not unique. Middle: the vertices of $\mathcal{S}_3$ are not defined only by the endpoints of s_3. Thin red segments show the last link of $\pi_3(\cdot)$. The figure is obtained experimentally. Right: there is no unique point x' (reflection of x in s) such that the length of the shortest tour from x, reflecting off s, to any point y, would equal $|x'y|$, and such that the point of contact of the tour with s would be, similarly to the L_2 case, that of the intersection of s and $x'y$.

$O(k)$ breakpoints. Thus, the *full combinatorial shortest path map* has linear complexity (this is in contrast with the L_2 case, when the map is shown to have exponential size in the worst case [47]). Finally, We give an algorithm to build ℓ_k and $\mathcal{S}_k$ for $k = 1 \dots K$ in total $O(K^2)$ time. The subdivisions $\mathcal{S}_1, \dots, \mathcal{S}_k$, of total linear size, allow to answer the queries for the shortest k-path to any point in the plane in $O(k)$ time.

Structure of $\mathcal{S}_k$

In this section we show that $\mathcal{S}_k$ has constant complexity.

Lemma 7.1. *The problem of finding the shortest k-path $\pi_k(z)$ to a point $z \in \mathbb{R}^2$ may be formulated as a linear program.*

Proof. Consider the following program for finding $\pi_k(z)$:

$$\begin{aligned} \text{minimize} \quad & t_1 + \dots + t_{k+1} \\ \text{subject to:} \quad & t_i \geq ||p_i - p_{i-1}||_1 \quad i = 1 \dots k+1 && (3) \\ & p_i \in s_i \qquad\qquad i = 1 \dots k && (4) \\ & p_0 = s, \quad p_{k+1} = z \end{aligned}$$

The decision variables of the program are the coordinates of the first contact points $p_1, \ldots, p_k$. The constraints (4) are linear. Each of the constraints (3) may be written as a set of five linear constraints:

$$t_i = ||p_i - p_{i-1}||_1 \qquad \Leftrightarrow \qquad \begin{aligned} t_i^x &\geq p_i^x - p_{i-1}^x \\ t_i^x &\geq p_{i-1}^x - p_i^x \\ t_i &= t_i^x + t_i^y \\ t_i^y &\geq p_i^y - p_{i-1}^y \\ t_i^y &\geq p_{i-1}^y - p_i^y \end{aligned}$$

where p_i^x, p_i^y are the coordinates of p_i. Thus, the program is an LP. □

It is a well-known fact [16, Problem 6.70] that as the right-hand side of a minimization LP changes linearly with rate λ, the objective function is a piecewise-linear convex function of λ. Thus,

Lemma 7.2. *$\ell_k(p)$ is a convex function of p.*

Let $z \in \mathbb{R}^2$ be a point in the plane.

Lemma 7.3. *The distance $|zp|$ from z to a point $p \in s_k$ is a convex function of p.*

Lemma 7.4. *$p_k(z)$ is a contiguous subset of s_k.*

Proof. Consider $|\pi_k(z)| = |zp| + |\pi_{k-1}(p)| = |zp| + \ell_k(p)$ as a function of $p \in s_k$. From Lemmas 7.2 and 7.3 it is a convex function and thus, its minimizers form a contiguous subset of its domain. □

We now describe the structure of $\mathcal{S}_k$. Some points in the plane may be readily assigned to vertex cells. Indeed, for a point $x \in \mathbb{R}^2$ let $I(x), II(x), III(x), IV(x)$ be the quadrants of the coordinate system with the origin at x. Lets say that the quadrants $I(x)$ and $III(x)$ are *opposite* each other; similarly, say that the quadrants $II(x)$ and $IV(x)$ are opposite. Consider now the coordinate system with the origin at a_k. Since s_k is not horizontal/vertical, s_k fully lies in one of the quadrants $I(a_k)$, $II(a_k)$, $III(a_k)$, or $IV(a_k)$. Let $Q(a_k)$ be this quadrant; let $Q'(a_k)$ be the quadrant, opposite $Q(a_k)$ (Fig. 37). The tours to the points in the quadrant $Q'(a_k)$ may without loss of generality go through a_k. Similarly, the tours to the points in the quadrant $Q'(b_k)$ may without loss of generality go through b_k. Thus,

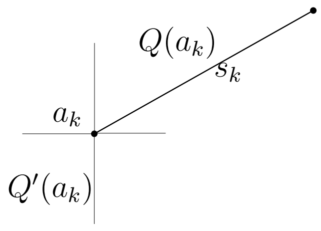


Figure 37: $s_k \subset Q(a_k)$.

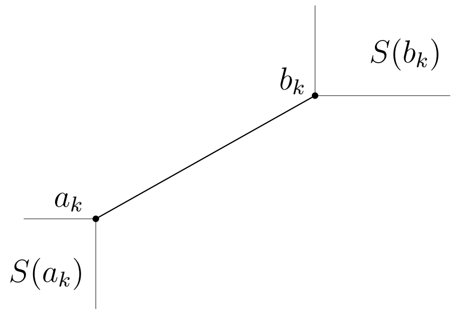


Figure 38: For $z \in S(a_k)$, $a_k \in p_k(z)$; for $z \in S(b_k)$, $b_k \in p_k(z)$.

Lemma 7.5. *The points in $Q'(a_k)$ belong to the vertex cell $S(a_k)$ of a_k. The points in $Q'(b_k)$ belong to the vertex cell $S(b_k)$ of b_k (Fig. 38).*

The points in the plane that are not assigned to the vertex cells form an "hourglass" with s_k as the "neck" (Fig. 39). Call the two "bulbs" of the hourglass, A_k and B_k, the *sides* of s_k (so that $\forall x, y \in \mathbb{R}^2 \setminus (s_k \cup S(a_k) \cup S(b_k))$, $xy \cap s_k = \emptyset$ if and only if x and y are on the same side of s_k). By Lemma 7.4 and continuity, the shortest $(k-1)$-paths to all points in s_k may arrive, without loss of generality, from the same side of s_k:

Lemma 7.6. *Either the shortest $(k-1)$-paths to any point on s_k arrive from A_k, or the shortest $(k-1)$-paths to any point on s_k arrive from B_k.*

Thus, k-paths to the points in one of the sides reflect off s_k, while k-paths to the points in the other side pass through the segment; to understand which side is which it is enough to compute the path $\pi_{k-1}(p)$ for an arbitrary point p interior to s_k. We define the *pass-through cell* $\mathcal{P}_k$ of $\mathcal{S}_k$ to be that side of s_k, k-paths to points in which pass through s_k.

Suppose, without loss of generality, that $\mathcal{P}_k = A_k$ (Fig. 40). Now the only missing part in the subdivision $\mathcal{S}_k$ is the assignment of the points in B_k. We

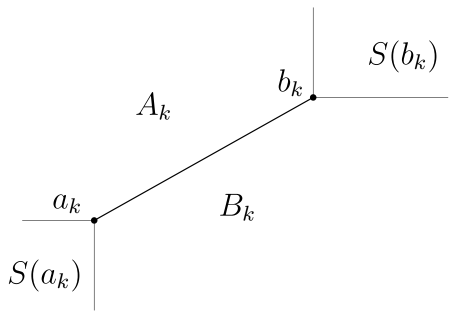


Figure 39: The hourglass.

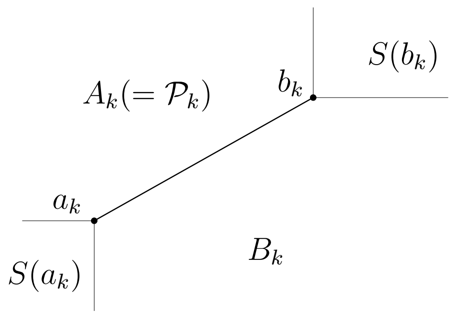


Figure 40: Vertex cells $S(a_k)$ and $S(b_k)$ and the pass-through cell $\mathcal{P}_k$ are known now.

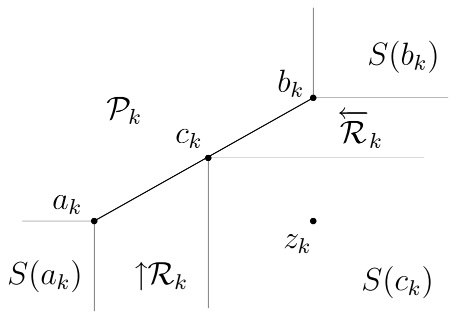


Figure 41: The cells of $\mathcal{S}_k$. $\forall z \in S(c_k), p_k(z) = c_k$.

show in Lemma 7.7 below that $\mathcal{S}_k$ has at most three cells in B_k. Specifically (Fig. 41),

1. For any point z in one of the cells we have $p_k(z) = c_k$, where c_k is some point in s_k (see proof of Lemma 7.7 for the definition of c_k). In other words, shortest k-paths to any point in the cell go through the same point, c_k, of s_k. This cell is denoted $S(c_k)$ in Fig. 41.

2. The last link of $\pi_k(z)$ to any point z in the second cell is horizontal. This cell is denoted $\overleftarrow{\mathcal{R}}_k$ in Fig. 41.

3. The last link of $\pi_k(z)$ to any point z in the third cell is vertical. This cell is denoted $\uparrow\mathcal{R}_k$ in Fig. 41.

Lemma 7.7. *$\mathcal{S}_k$ has at most three cells in B_k as described above.*

Proof. Let $\mathcal{T}_k = R_k \cap B_k$ be the right triangle $z_k a_k b_k$ (Fig. 42). Since the k-paths to any point in $B_k \setminus \mathcal{T}_k$ must go through $\mathcal{T}_k$, it is enough to consider only the restriction of $\mathcal{S}_k$ to $\mathcal{T}_k$. Clearly, k-paths to a point $z \in \mathcal{T}_k$ may go through a point on s_k lying between $v_k(z)$ and $h_k(z)$ (see Fig. 42).

Assume that $p_k(z_k)$ is a single point $c_k \in s_k$ (if it is not so, let c_k be an arbitrary point in $p_k(z_k)$). Let R'_k be the axis-aligned rectangle having z_k and c_k as opposite corners (Fig. 43). Consider a point z in R'_k. The path $z_k p_k(z_k) = z_k c_k$ may without loss of generality go through z. Thus, $p_k(z) = c_k$ for otherwise, the k-paths to z_k would rather not go through c_k either. Since z was an arbitrary point in R'_k, any point $z \in R'_k$ has c_k as their first contact point with s_k. Hence, shortest k-paths to all points in $S(c_k)$ (Fig. 41) have c_k as their first contact point with s_k.

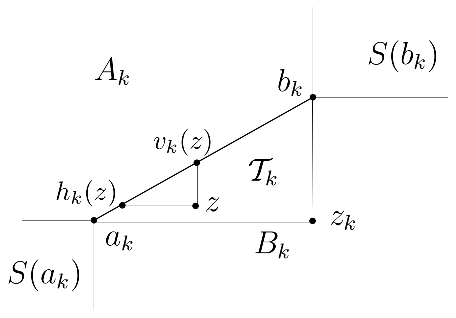


Figure 42: $\mathcal{T}_k = R_k \cap B_k$ so that $a_k = v_k(z_k)$, $b_k = h(z_k)$, $z_k \in B_k$. For $z \in \mathcal{T}_k$, without loss of generality $p_k(z) \in h_k(z)v_k(z)$.

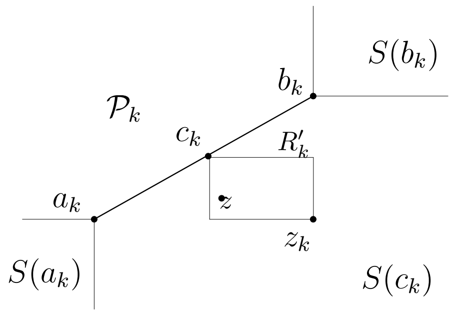


Figure 43: $c_k = p_k(z_k)$.

The remaining part of B_k, $B_k \setminus S(c_k)$, consists of two regions. By continuity, for any point z in one of them, $p_k(z) = v_k(z)$, in the other — $p_k(z) = h_k(z)$. Refer to Fig. 41. □

Remark. It may happen that one (or both) of the cells $\uparrow\mathcal{R}_k$, $\overleftarrow{\mathcal{R}}_k$ is empty, if, say, $c_k = a_k$; in this case the vertex cell $S(a_k)$ is a halfplane through a_k.

Building $\mathcal{S}_k$

As Dror et al. [47] did for the case of L_2-shortest tours, we build the subdivision $\mathcal{S}_k$ from the subdivisions $\mathcal{S}_1 \dots \mathcal{S}_{k-1}$. First we assign $S(a_k)$ and $S(b_k)$. Then we determine which cells of $\mathcal{S}_{k-1}$ are intersected by s_k. For a point p in the part of s_k, intersecting a vertex cell $S_{k-1}(v)$ of $\mathcal{S}_{k-1}$ (where v is a_{k-1} or b_{k-1}), $\ell_k(p) = |pv| + \ell_{k-1}(v)$. For p in a reflect cell, say, $\overleftarrow{\mathcal{R}}_{k-1}$, $\ell_k(p) = |ph_{k-1}(p)| + \ell_{k-1}(h_{k-1}(p))$. For $p \in s_k \cap \mathcal{P}_{k-1}$, we look at the subdivision $\mathcal{S}_{k-2}$ to deduce $\ell_k(p)$. Finally, we find $c_k \equiv p_k(z_k) =$

$$\arg \min_{p \in s_k \cap (\mathbb{R}^2 \setminus (S(a_k) \cup S(b_k) \cup \mathcal{P}_k))} (\, |z_k p| + \ell_{k-1}(p) \,)$$

to complete the subdivision.

The time-dominating step of the above procedure is computing the function $\ell_k(p)$. Since, in principle, part of s_k may fall into $\mathcal{P}_{k-1}$, the complexity C_k of $\ell_k(p)$ may be at least $C_{k-1} + C_{k-2}$, which is at least exponential in k. We show (Lemma 7.10) that in fact, C_k is linear and thus, $\mathcal{S}_k$ and $\ell_k(p)$ can be built in $O(k)$ time.

Lemma 7.8. *Suppose that a point z moves from a_k to b_k along the segment s_k. Then $p_{k-1}(z)$ moves (weakly) monotonically along a subsegment of s_{k-1}.*

Proof. Since the cells of $\mathcal{S}_{k-1}, \dots, \mathcal{S}_1$ are convex, once z leaves a cell, it never enters it again. While z is within a reflect cell, $p_{k-1}(z)$ moves together with z (horizontally or vertically). While z is within a vertex cell, $p_{k-1}(z)$ does not move at all. □

Let $p_1^*, \dots, p_{k-1}^*$ be the first points of contact of the path $\pi_k(z)$ with the segments $s_1, \dots, s_{k-1}$: $p_{k-1}^* = p_{k-1}(z)$, $p_{k-2}^* = p_{k-2}(p_{k-1}(z))$, $p_{k-3}^* = p_{k-3}(p_{k-2}^*) = p_{k-3}(p_{k-2}(p_{k-1}(z)))$, $\dots$, $p_1^* = p_1(p_2^*)$.

Lemma 7.9. (Monotonicity Lemma). *As z moves along s_k, the first points of contact $p_1^*, \dots, p_{k-1}^*$ move (weakly) monotonically along corresponding segments.*

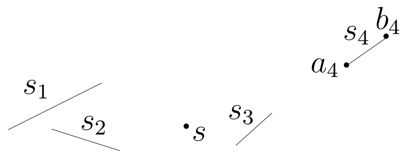


Figure 44: Going to any point in the plane visiting s_4 after visiting s_1, s_2, s_3 may without loss of generality be done by going through a_4; hence $\mathcal{S}_4$ consists of just one cell $S(a_4) = \mathbb{R}^2$.

Proof. By Lemma 7.8 and induction. □

Lemma 7.10. $C_k \leq 3(k-1)$.

Proof. The breakpoints of $\ell_k(p)$ correspond to events when one of the first points of contact hits a vertex (a_k, b_k, or c_k) of the corresponding subdivision. □

Remark. We believe that there exist problem instances achieving $C_k = \Omega(k)$.

Remark. It can happen that $\mathcal{S}_k$ consists from only one cell, i.e., $C_k = 1$ (Fig. 44).

Remark. The *full combinatorial shortest path map* is a subdivision of the plane into cells such that the shortest K-paths to any point within a cell is in the same-type contact with *all* K segments; in L_2, its complexity is exponential in K [47]. An important corollary of Lemma 7.10 is

Corollary 7.11. *The complexity of the full combinatorial shortest path map is $O(K)$.*

This is in contrast with L_2 [47].

Finding the Optimal Tour

Given the subdivisions $\mathcal{S}_1, \ldots, \mathcal{S}_K$ we can readily compute, for any $k = 1 \ldots K$, the optimal tour $\pi_k(z)$ to any point $z \in \mathbb{R}^2$. We start by locating z in $\mathcal{S}_k$. If z is in the pass-through cell $\mathcal{P}_k$ of $\mathcal{S}_k$, then $\pi_k(z) = \pi_{k-1}(z)$ and we continue by locating z in $\mathcal{S}_{k-1}$. If z is in a vertex cell $S_k(v)$ of $\mathcal{S}_k$, then $p_k(z) = v$. If z is in a reflect cell, we follow the "arrow" of the cell until hitting the boundary of s_k at $p_k(z)$. We then recursively compute $\pi_{k-1}(p_k(z))$.

Theorem 7.12. *The subdivisions $\mathcal{S}_1, \ldots, \mathcal{S}_K$ of the plane, of total size $O(K)$, can be built time $O(K^2)$ that enable computing shortest tours $\pi_k(z)$ to a query point z in $O(k)$ time.*

7.1.2 Second-Order Conic Programming Solution

In this section we study the problem of finding a shortest tour visiting a given sequence of convex bodies in $\mathbb{R}^d$. This is an attempt to attack the touring problem in its full generality: we investigate high-dimensional cases ($d \geq 2$); we consider convex bodies bounded by (hyper)planes and/or (hyper)spheres; we do not restrict the start and the goal positions of the tour to be single points, we measure the length of the tour according to either Euclidean or L_1 metric. Formulating the problem as a second-order cone program (SOCP) makes it possible to incorporate distance constraints, which cannot be handled by a purely geometric algorithm.

We implemented the SOCP in MATLAB and obtained its solution with the SeDuMi package. We ran computational experiments, which suggest that the proposed solution is practical.

Second-Order Conic Programming (SOCP) SOCP is known to be applicable to a number of computational geometry problems, such as finding extremal volume ellipsoids, centering, separation and classification, placement and facility location, projection and distance problems, intersection and containment of polyhedra, floor planning [22], architectural design [102]; see also [126]. SOCP also provides a natural framework to attack geometric problems in which the goal is to optimize the length of a network (embedding of a planar graph), possibly, under linear and quadratic constraints. A classical example is the Weber (Facility Location) problem [46, 93, 124]: the total length of a *star* is minimized, when the locations of degree-1 nodes of the star are given. Here we use SOCP to minimize the total length of a *path*, when the nodes of the path are constrained to stay within convex regions. SOCP formulation also allows one to incorporate certain length and distance constraints. A framework similar to ours is outlined in [22, page 433], where it is applied to placement and location problems.

SOCP Formulation

We assume that each body $\mathcal{B}_i\,(i = 1 \dots K)$ in the sequence is given as the intersection of a set of J_i bounding (hyper)halfspaces and (hyper)spheres: $\mathcal{B}_i = \{x \in \mathbb{R}^d \,|\, x \in \mathcal{H}_{ij}^-, j = 1 \dots J_i\}$. Each of the bounding surfaces gives rise to a linear or conic constraint $x_i \in \mathcal{H}_{ij}^-$, where x_i is the i-th vertex of the path.

Then the touring problem may be formulated as the following SOCP:

$$\begin{array}{rll} \text{minimize} & t_1 + t_2 + ... + t_{K-1} & \\ \text{subject to:} & t_i \geq ||x_{i+1} - x_i|| & i = 1 \ldots K-1 \\ x_i \in \mathcal{H}_{ij}^- & i = 1 \ldots K & j = 1 \ldots J_i \end{array}$$

If the bodies in the sequence have in total n constraints, then the SOCP allows us to find an ϵ-approximate tour in $O\left(d^3 n^{1.5} K^2 \log \frac{1}{\epsilon}\right)$ ([90]).

Additional Constraints

In some applications it is natural to ask that the length of each link of the tour does not exceed a certain bound L_i. Sometimes, also a set $C = \{c_1, \ldots, c_M\}$ of M control points is given, with the requirement that (some of) the bends of the tour occur close to (some of) the control points: $||x_i - c_m|| \leq d_{im}$, where d_{im} ($i = 2 \ldots K-1$, $m = 1 \ldots M$) are some constants.

Imposing any of the above constraints makes it unlikely that the problem can be efficiently solved by purely geometric techniques, like the ones in [99] and subsequent papers on the Weighted Region Problem (see [98]). At the same time, these additional constraints are conic and thus naturally can be handled by our program.

L_1 metric

Our SOCP can also be applied to the touring problem when the length of the tour is measured according to the L_1 metric. It requires only a slight change in the SOCP: for each link of the tour one variable per dimension is introduced. In $\mathbb{R}^2$, e.g., the new SOCP will be:

$$\begin{array}{rll} \text{minimize} & t_1^x + t_1^y + ... + t_{K-1}^x + t_{K-1}^y & \\ \text{subject to:} & (x_i, y_i) \in \mathcal{H}_{ij}^- & j = 1 \ldots J_i \\ & t_i^x \geq ||x_{i+1} - x_i|| & i = 1 \ldots K-1 \\ & t_i^y \geq ||y_{i+1} - y_i|| & i = 1 \ldots K-1 \end{array}$$

Figure 45 shows the tours of a sequence of line segments, optimal under L_1 and L_2 metrics.

Weighted Links

It is straightforward to modify our solution so that it handles the weighted version, in which each link is assigned a weight. If $w_1, \ldots, w_{K-1}$ are the weights

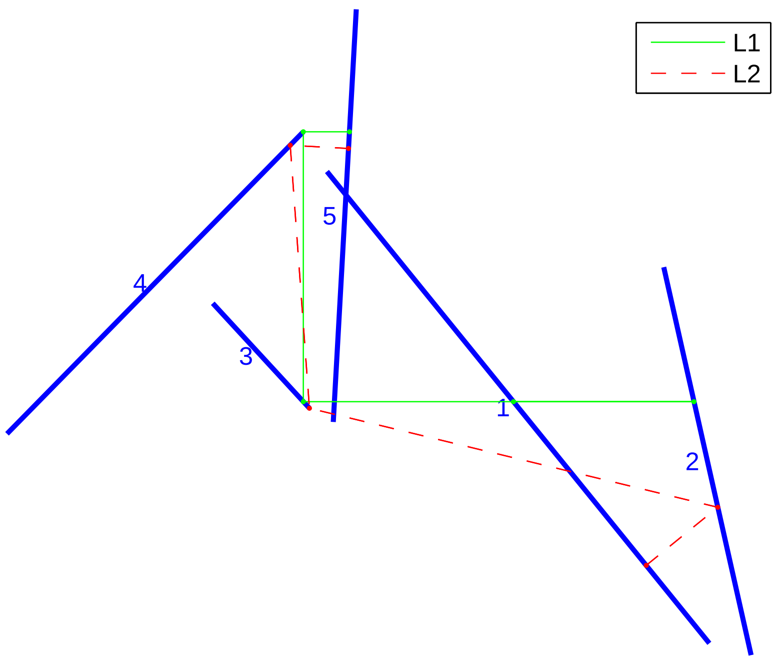


Figure 45: Tours, optimal in different metrics.

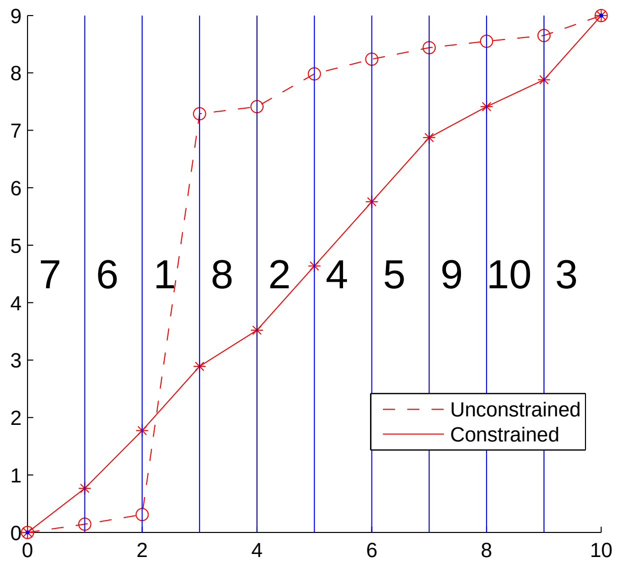


Figure 46: Illustration of two optimal paths through weighted strips, showing the constrained (solid) and unconstrained (dashed) optimal routes. Here, the length constraint is $L = 1.5$. The bold number in each strip is its weight.

of the links, the objective function changes to $\min w_1 t_1 + ... + w_{K-1} t_{K-1}$; the rest of the SOCP remains the same.

Figure 46 shows the optimal weighted tours of a sequence of parallel line segments. Without the constraint on the length of the links, the path obeys Snell's Law of Refraction; the behavior of the constrained path is different.

Computational Experiments

We implemented the described program in MATLAB. The solution to the SOCP was obtained with the SeDuMi package by Jos Sturm [118]. We report here the run times for the simplest case, when the bodies are parallel straight line segments of equal length – edges in a weighted subdivision of a box (see Fig. 46).

Theoretically, the running time of the algorithm is $O(K^{3.5} \log 1/\epsilon)$ to achieve accuracy ϵ. We did not change the default SeDuMi setting $\epsilon = 10^{-9}$ in our experiments. We were able to solve instances of the problem with K up to 5000. The SOCP algorithm performed about 15–25 iterations in every instance of

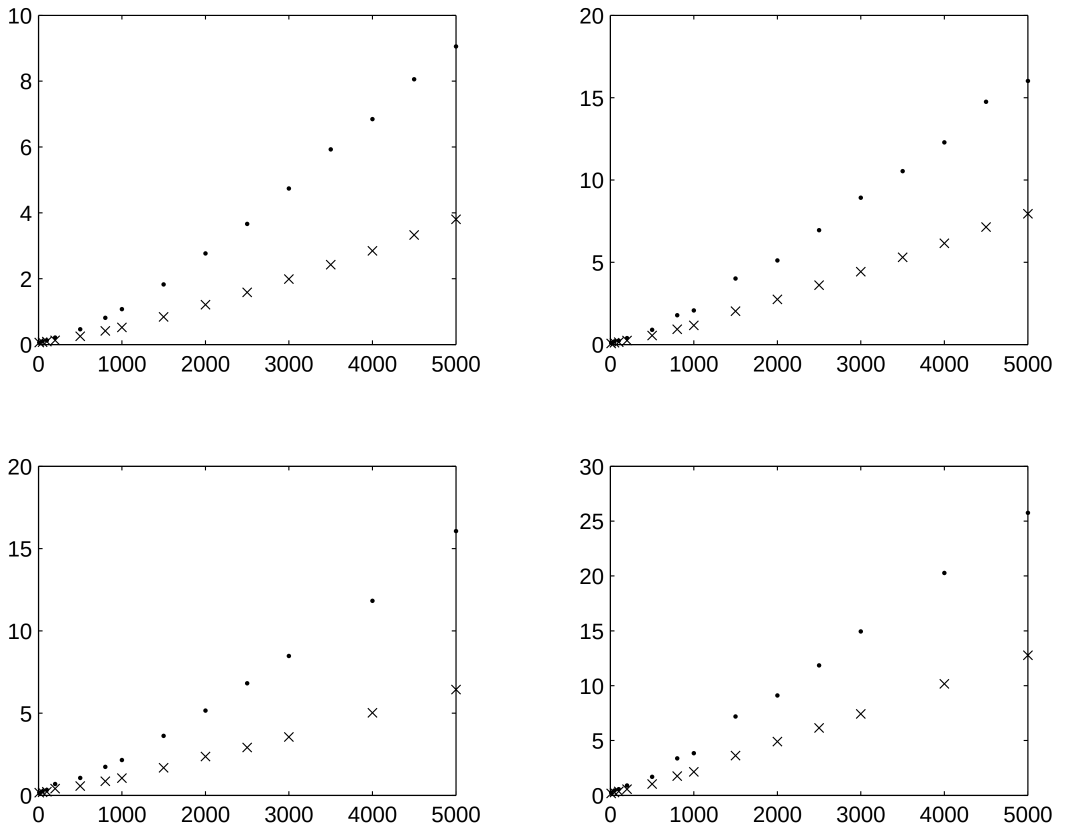


Figure 47: Running times, Windows Machines. Top: 1.9MHz, 512M RAM Compaq laptop; bottom: 1.7MHz, 256M RAM Dell desktop. Left: unconstrained; right: constrained. Dots – run time, sec; crosses – run time per iteration, .1 sec.

the problem. This coincides with the observation, made by Lobo et al. in [90] about primal-dual interior point method for SOCP: the typical number of iterations ranges between 5 and 50, almost independent of the problem size.

The average (over about 100 runs) actual running time of the algorithm for different problem sizes is presented in Figure 47.

Figures 48, 49, 50 and 51 show solutions of the general touring problems in 2 and 3 dimensions.

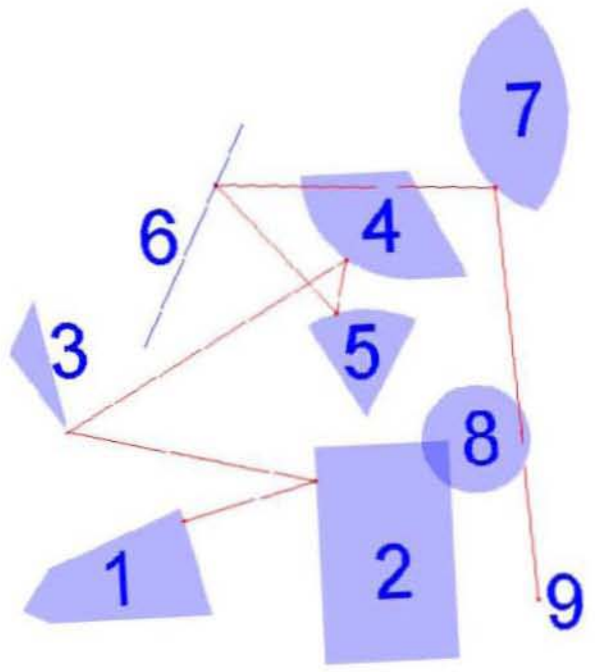


Figure 48: A path in 2D.

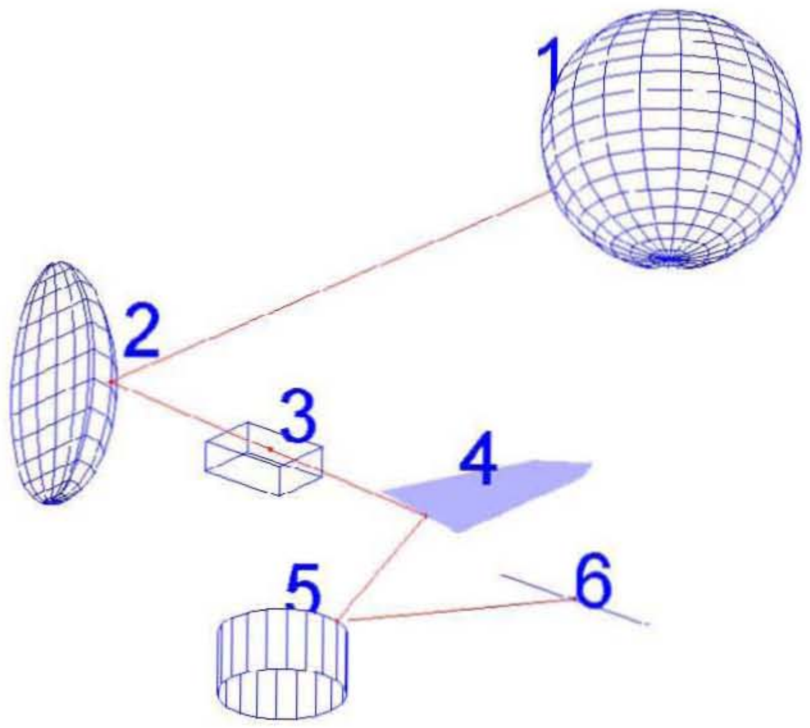


Figure 49: A path in 3D.

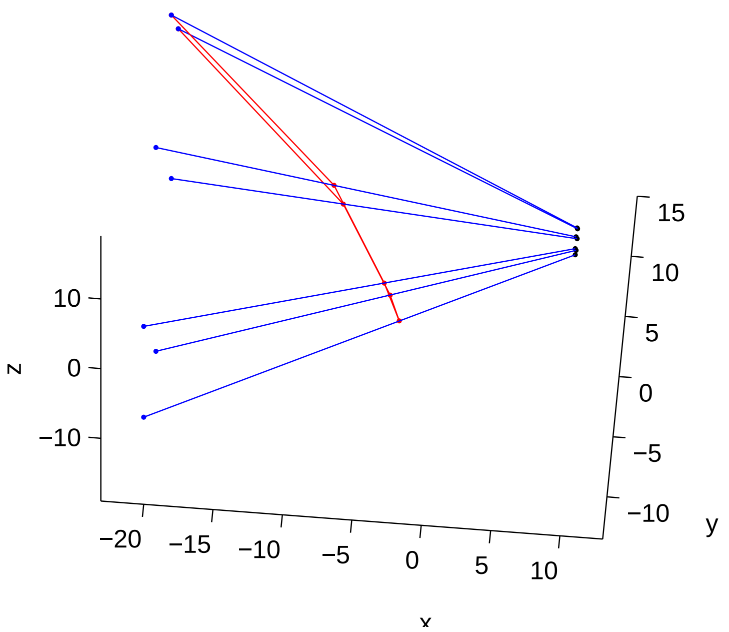


Figure 50: Touring lines in 3D; path endpoints are given.

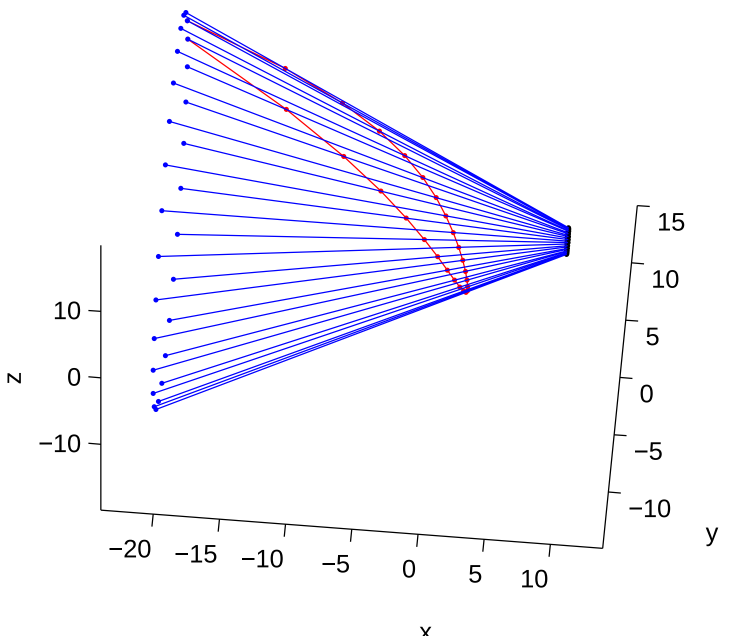


Figure 51: Touring lines in 3D; path endpoints are given.

7.1.3 Hardness Results

We complement our solutions with NP-hardness results, showing that the assumptions we make in the statement of the problem are crucial to the efficient solvability.

First of all, if the order in which the bodies are to be visited is not given, then our touring problem becomes TSP with neighborhoods (see [98, Chapter 7.4]) and thus is NP-hard.

If the bodies in the sequence are not convex, then our problem is NP-hard by the reduction presented in [47].

Finally, if the length of each link of the tour is bounded from *below*, then even the simplest version of our problem is weakly NP-hard. Indeed, consider the problem of finding a shortest path visiting a sequence of parallel line segments of equal length, with a given start point s and goal point t.

We use a reduction from PARTITION: Given a set $A = \{a_1, \dots, a_K\}$ of K integers summing to S, is there a subset $A' \subset A$ of elements summing to $S/2$? Given an instance of PARTITION, we construct an instance of the touring problem such that the PARTITION problem has answer "Yes" if and only if the optimal path has length at most $(K+1)L$, where L the lower bound on the length of the links (common for all links). See Figure 52.

Theorem 7.13. *The optimal touring problem is weakly NP-hard if a lower bound is specified for the length of each segment of the path.*

7.2 Shortest Rectilinear k-Link Path

In this section we present an algorithm for computing k-link rectilinear shortest paths among rectilinear obstacles in the plane. We extend the "continuous Dijkstra" paradigm to store the link distance information associated with each propagating "wavefront". Our algorithm runs in time $O(kn \log^2 n)$ and space $O(kn)$, where n is the number of vertices of the obstacles. Previous algorithms for the problem had worst-case time complexity $O(kn^2)$.

Our algorithm builds a j-link shortest path map, rooted at a given source s, for each $j \leq k$. A shortest path query from s to a query point t can then be answered in time $O(\log n + j)$.

We consider the bi-criteria rectilinear two-dimensional shortest path problem: Determine a rectilinear path of minimum (L_1) length, having at most k links, from s to t that avoids the interiors of a set of disjoint simple rectilinear obstacles having a total of n vertices. The bi-criteria rectilinear path

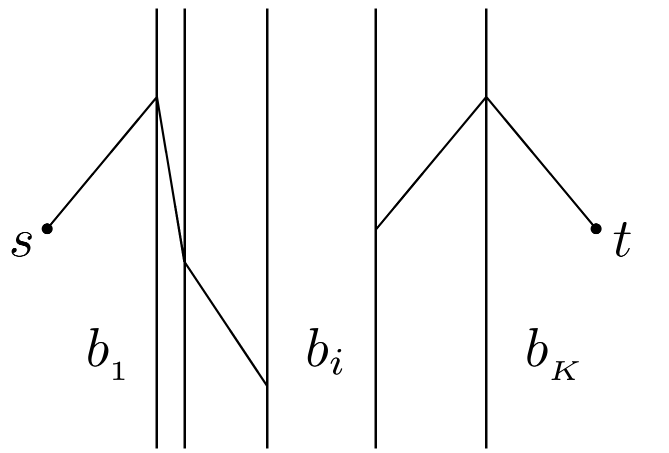


Figure 52: The hardness reduction. The width of strip i is $b_i = \sqrt{L - a_i^2}$. If the ith link has positive slope, then a_i is chosen to be in the subset A'.

problem naturally arises in certain wire-routing applications in which one is interested in finding a shortest rectilinear path for a wire that avoids a given set of components and is constrained to have at most k links.

Bi-criteria path problems have received considerable attention in the computational geometry literature; see, e.g., [97, 98, 108]. Many problems are known to be NP-hard [12], but often only weakly so. For the special case of two criteria consisting of *length* (e.g., L_1 or L_2) and *link distance*, several polynomial-time algorithms are known for basic geometric optimal paths. See [87] for a survey specific to rectilinear paths and rectilinear obstacles, as studied here.

It is challenging to compute Euclidean shortest paths having at most k links, since there is no simple discrete graph that is "path preserving" for optimal paths. The special case of k-link paths in simple polygons and some approximation algorithms for more general cases are considered in [100].

In rectilinear polygonal domains, efficient algorithms are known for the bi-criteria path problem that combines *rectilinear* link distance and L_1 length. One can achieve worst-case time $O(kn^2)$ (more precisely, $O(k(m + n \log n))$, where m is a number of crossings in an arrangement, and is worst-case $\Theta(n^2)$); see [87, 127]. More efficient algorithms, running in nearly linear time, are known for optimal paths in a "combined metric"; see [28, 128][12]. Results are

[12]Some of these nearly-linear time results are said to apply to optimizing any nondecreasing function $f(l, j)$ of the L_1 length l and the number of links j; however, we suspect there is

also known in higher dimensions for optimal paths in a combined metric, if the obstacles are given as a set of axis-parallel boxes [36].

Our results give an $O(kn \log^2 n)$ time algorithm, improving the $O(kn^2)$ bound by roughly a factor of n. We construct a family of planar subdivisions (link-restricted shortest path maps), one for each $j = 1, \ldots, k$, which gives a decomposition of the plane into cells according to the combinatorial type of a j-link shortest rectilinear path from the source s. For any query point t, the length of a shortest j-link path from s to t is determined by locating t in the jth map.

Overview of the Algorithm

We apply the "continuous Dijkstra" paradigm [94], which has been applied successfully to solving many optimal path problems in geometry. Since our new algorithm is based on a variant of the continuous Dijkstra algorithm of [96], we begin with a review of that method and then describe the changes necessary to extend it to our problem.

The algorithm considers the effects of sweeping an advancing "wavefront" from a source point s to all points of free space $\mathcal{F}$. (The *wavefront* at distance D is the set of points p of $\mathcal{F}$ for which the shortest path length from s to p is D.) In order to simulate the advancement of wavefronts correctly, the following information associated with each segment $\overline{qq'}$ of the wavefront is stored in a priority queue (called the *event queue*):

(a). its orientation, which will always be either northwest (NW), southwest (SW), southeast (SE) or northeast (NE) in the case of the L_1 metric;

(b). its endpoints q and q', which are the positions of the segment's endpoints at the moment the segment is first instantiated, before it starts being "dragged";

(c). its left and right *track rays* — these are the rays along which q and q' must be dragged, and they may be horizontal or vertical rays through free space or rays containing obstacle edges;

(d). the *stop points* L and R of the left and right track rays — these are the first obstacle points "hit" by the left and right track rays. (If the track

a misunderstanding, since one could seemingly apply this result to the function $f(l, j)$ that is l if $j \leq k$ and ∞ otherwise, giving a nearly-linear time algorithm (independent of k) for the k-link shortest rectilinear path. The algorithms are based on applying Dijkstra's algorithm in a single-criterion weighted graph, rather than a dynamic program (e.g., Bellman-Ford) that searches for shortest j-link paths.

rays intersect each other at point u before they hit obstacles, then $L = R = u$, where u is the inside corner of the corresponding segment dragging query.);

(e). its *root* r, which is an obstacle vertex that is responsible for propagating the portion of the wavefront to which the segment belongs;

(f). its *contact list*, which is the set of obstacle edges that the dragged segment touches (including the obstacle edges on which its endpoints may be sliding);

(g). its *event position* $\overline{q_e q_e'}$, which is the next position of the segment at which the contact list changes;

(h). its *event point* p, which is the point that is responsible for the change in the contact list when the segment reaches its event position. The event point p must lie on the boundary of an obstacle, and it will either be a stop point or a vertex;

(i). the *event distance*, which is the distance from s at which the event point is encountered by the segment.

The segments in the event queue are ordered according to their event distances. The *next event* is the dragged segment whose event distance is minimum and is obtained by popping the queue. In the case of ties, we can order the event distances by the lexicographic ordering of the x- and y-coordinates of their roots.

Each obstacle vertex u has associated with it a sorted list, the *SE-hit list*, $\mathcal{R}^{SE}(u) = \{r_1, \ldots, r_N\}$ of roots r_i of dragged segments that are southeast of u and are such that the dragged segment has "hit" point u (i.e., u has been an event point for a dragged segment rooted at r_i, and this event has already occurred). Similar definitions apply to the hit lists $\mathcal{R}^{NE}(u)$, $\mathcal{R}^{NW}(u)$, and $\mathcal{R}^{SW}(u)$ of the roots of the segments which have hit u from southwest, southeast and northeast respectively. The total size of all lists is bounded above by $O(n \log n)$.

Also associated with each obstacle vertex u is a *permanent label*, $d(u)$, which, at the conclusion of the algorithm, gives the length $\ell(s, u)$ of shortest path from s to u. Initially, $d(u) = +\infty$ for all u. We say that u has been *permanently labeled* if $d(u) < +\infty$. We say that a *non*-vertex point x has been permanently labeled if it lies in the region swept out by some dragged segment. Each vertex u also has a pointer, parent(u), which, at the conclusion of the algorithm, points to the parent of u in the shortest path tree SPT(s). Initially, parent(u) =NIL.

There are three types of events:

(I). the event point p is one of the stop points;

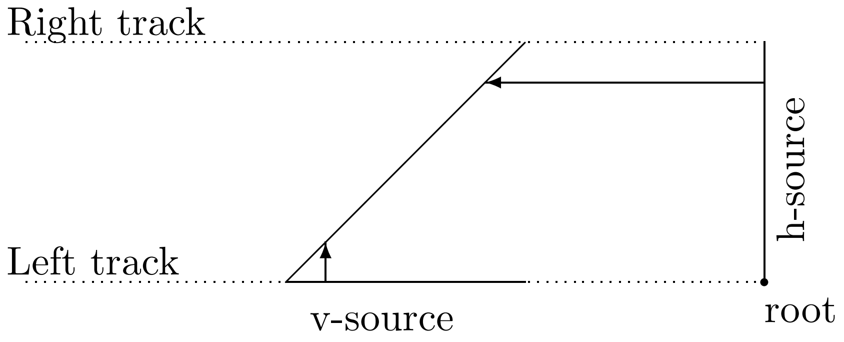


Figure 53: The information associated with a wavefront segment.

(II). p is interior to the dragged segment in its event position; and,

(III). p is a vertex encountered by an endpoint of the dragged segment.

The event queue is updated at each event so as to simulate the wavefront propagation correctly. Determining events in the continuous Dijkstra method involves answering *segment dragging queries* of special forms; see [96] for details.

Modifications to account for link distance In order to modify the above algorithm for the k-link path problem, we extend the continuous Dijkstra algorithm to store the (rectilinear) link distance from s to any point u on the wavefront. In particular, we distinguish between s-u paths ending with a vertical link and s-u paths ending with a horizontal link. To this end we associate with each wavefront segment one or two additional segments: a horizontal segment, called a *v-source* and/or a vertical segment, called an *h-source*. With each v-source v we store its link number, $l.v$, which is the link distance from s to v, and pointer to a predecessor h-source, $pred.v$. Then the shortest $(l.v + 1)$-link s-u path with last link vertical may go from s through $pred.v$ to v and then to u. We store similar information with each of the h-sources. Refer to Figure 53.

Each obstacle vertex u has now associated with it k SE-hit lists. For $j \in \{1 \dots k\}$, the hit list ${}^{j}\mathcal{R}^{SE}(u)$ contains the roots and sources of dragged segments that are southeast of u and are such that the dragged segment has "hit" point u *and* the link distance from s to the source is less than j. (Clearly, ${}^{1}\mathcal{R}^{SE}(u) \subseteq {}^{2}\mathcal{R}^{SE}(u) \subseteq \cdots \subseteq {}^{k}\mathcal{R}^{SE}(u)$, so we only store the corresponding set differences). We similarly define hit lists for other hit directions.

Initially, s is permanently labeled with 0. Four dragged segments rooted at s are inserted along with their distance labels into the event queue: NE, NW, SW and SE segments with the tracks being horizontal and vertical rays from s.

Events The propagation of the wavefront involves doing different things depending on whether the next event is of Type I, II, or III and on whether or not the event point p has already been permanently labeled. The cases are illustrated in Figures 54, 55, 56 and 57 for a segment dragged northeast. Processing the events for the segments propagating in other directions is similar. The details of the events processing are mostly the same as in [96]. It is important, though, to modify the clipping of wavefronts in order that the only wavefronts that are permitted to continue (and not be clipped appropriately) are those corresponding to Pareto-optimal solution paths.

Complexity of the algorithm Since we are propagating up to k different wavefronts (corresponding to up to k different link distances to the points on wavefronts), the complexity of the algorithm in [96] goes up by a factor of k and becomes $O(kn \log^2 n)$.

The proof of correctness is based on an induction argument, establishing that each point t reached (swept over) by the ith event (with associated L_1 distance d_i) have been reached by a dragged segment corresponding to each of the link distances $j = d_L(s,t), \ldots, l(t)$, where $d_L(s,t)$ is the rectilinear link distance from s to t and $l(t)$ is the (maximum) rectilinear link length of a shortest L_1 path from s to t.

7.3 The Box Mover Problem

In this section we show that the *optimization* problem is NP-hard for a wide class of motion planning puzzles, including classical SOKOBAN. We investigate a new problem, the Box Mover Problem (BMP), in which the agent is allowed to lift and carry boxes on a rectilinear grid in order to rearrange them. Some classical motion planning puzzles are special cases of BMP. We also identify a natural class of BMP instances, for which optimization is in NP, making the optimization problems from the class NP-complete.

There is a number of motion planning puzzles in which, given an arrangement of unit blocks (boxes) in the plane, one has to rearrange the boxes into another configuration by operating a robot which moves in the same plane amid the boxes. The classical example is SOKOBAN. ([43] provides a thorough description of the puzzles and corresponding algorithmic results.) The

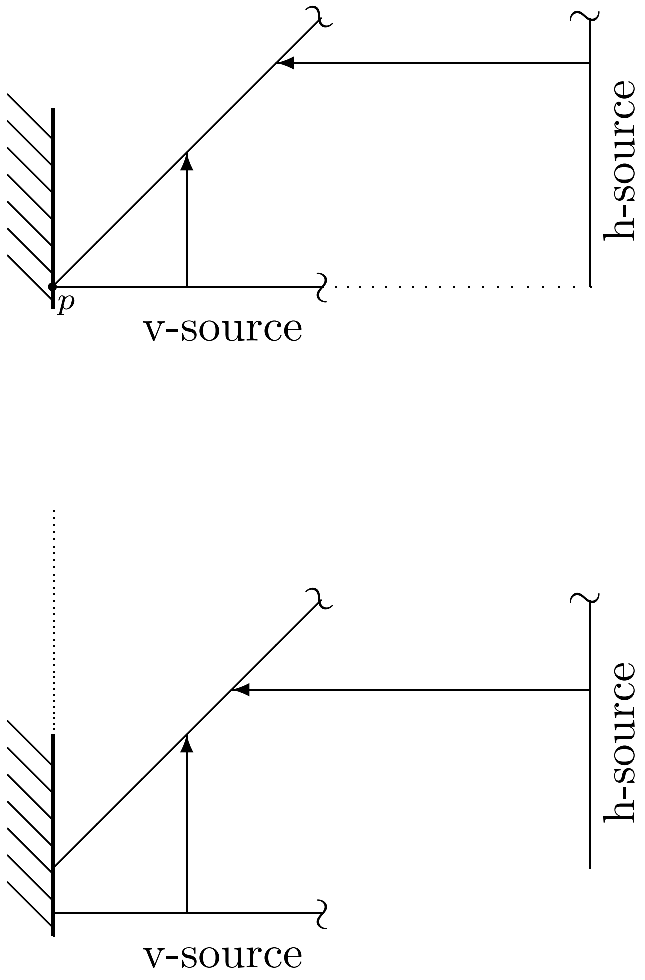


Figure 54: Type I event.

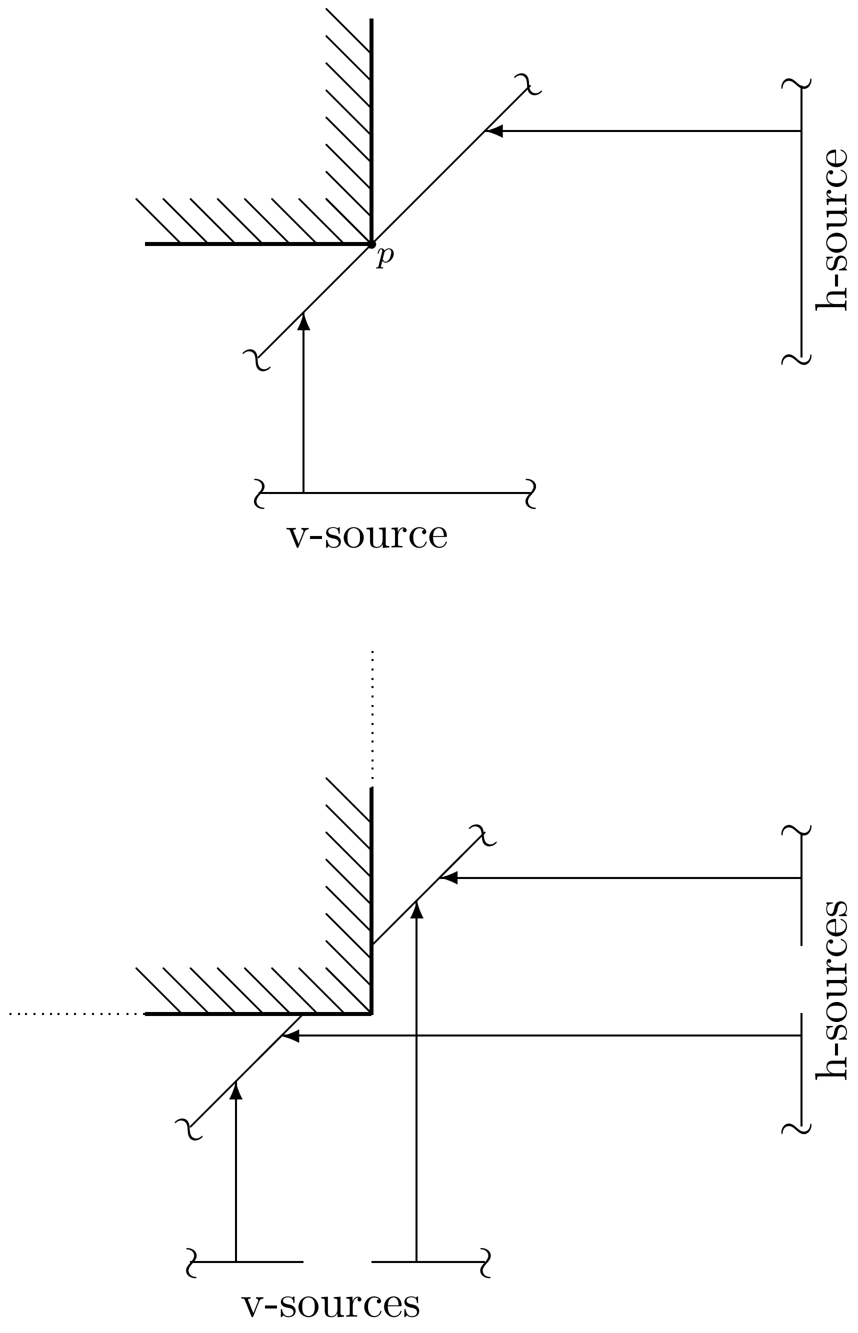


Figure 55: Type II event.

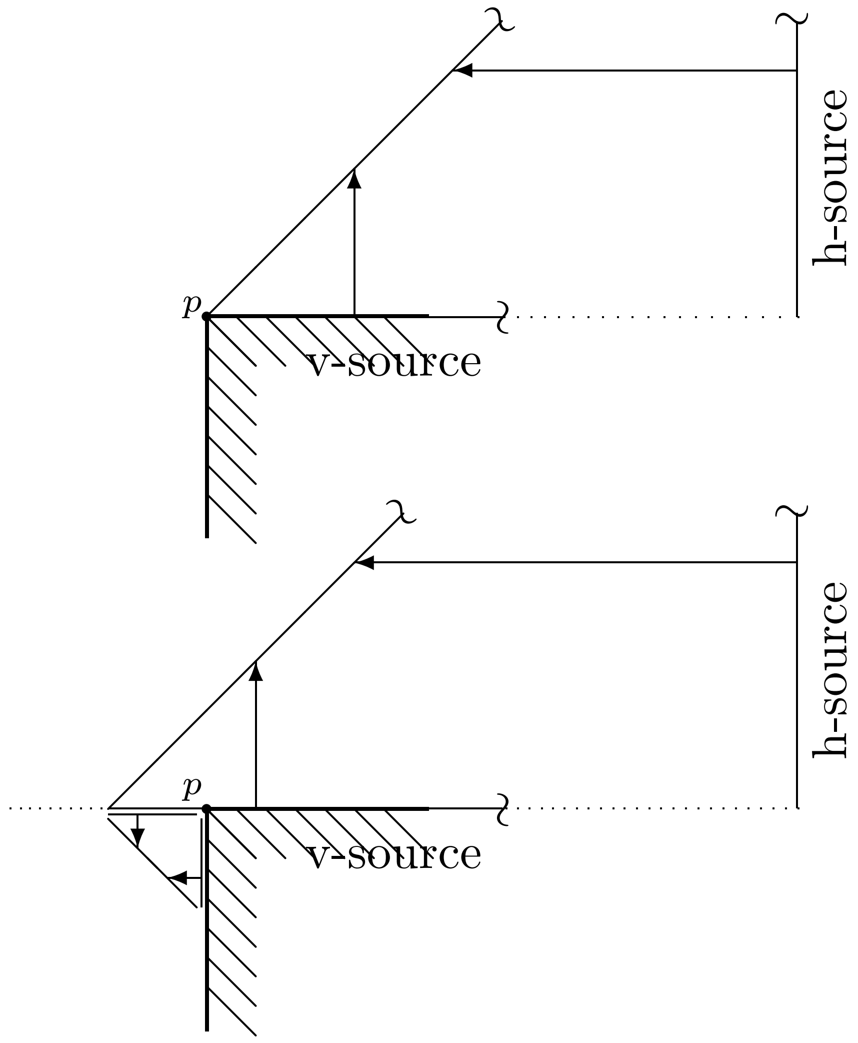


Figure 56: Type III event.

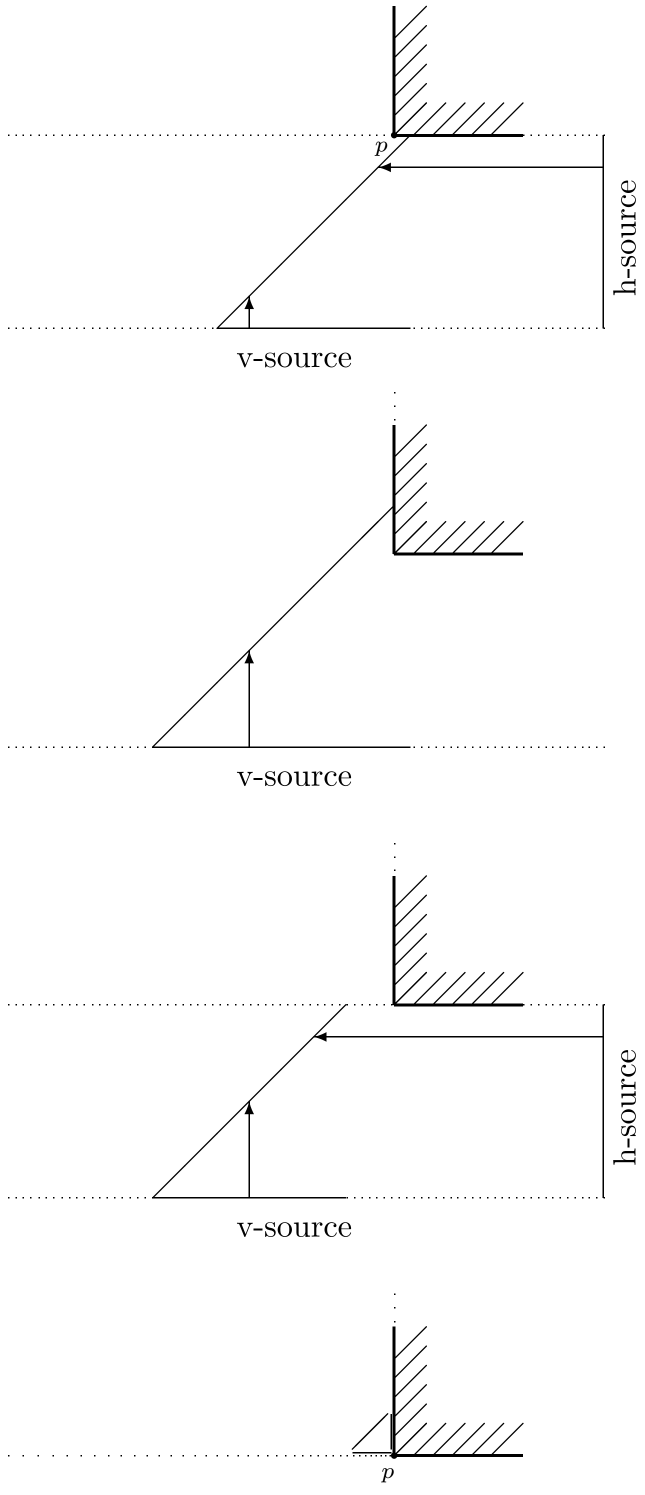


Figure 57: Another Type III event.

puzzles we consider here may be classified according to the following characteristics (the classification is adapted from [38]):

1. How powerful the robot is:
   - Can the robot push the boxes? How many at a time?
   - Can the robot pull the boxes? How many?
   - We introduce a “new dimension” for the robot: can the robot lift a box and put it to an adjacent position (including the position, occupied by the robot before the lift)? The new problem is dubbed the *Box Mover Problem* (BMP).
2. Box types
   - Are all boxes movable, or are some fixed to the plane? In other words, are we working on the infinite plane, with nothing else but the boxes on it, or are we constrained to a floor bound by rigid walls?
3. Robot path
   - Is the solution path required to have no self-intersections?
   - Are we looking for a closed path for the robot?
4. Boxes' IDs (”15”-style)
   - Are the boxes and the target positions labeled? This may be important with respect to the final configuration of the boxes; in SOKOBAN any box can occupy any target position.

Following notation in [38, 39, 45], we call BMP(k, p, l) the Box Mover Problem for the robot capable of pushing k, pulling p and lifting l boxes at one time. If some boxes may be fixed to the plane, the problem is called BMP(k, p, l)-F. If only non-self-intersecting paths are allowed for the robot, the problem is called BMP(k, p, l)-X. We do not require the robot to return to its initial position; it can stop right after all the boxes are in their target positions. Finally, if the boxes and the target positions bear labels, the problem is called #BMP(k, p, l). Thus, e.g., BMP$(1, 0, 0)$-F is the original SOKOBAN game, BMP(∞, ∞, ∞) is the Omnipotent Robot Problem (they also have problems), BMP$(k, 0, 0)$, BMP$(\infty, 0, 0)$ and BMP$(1, 0, 0)$-X are the Push-k, Push-* and Push-X versions of Push (see [43]).

To clarify rules for lifting, we emphasize that the robot can essentially "go under" a box: it can approach the box, swap positions with it and then put the box back. Such an operation requires 2 lifts. The robot can also carry a box to another location. We think of such carrying as a sequence of lifts; the number of necessary lifts equals the distance traveled by the robot with the box.

It is possible to come up with other rules for lifting. With some adjustment, our results remain valid for other rules as well.

Comparison with Previous Work

1. To our knowledge, previous research concentrated on investigating hardness of the *feasibility* problems, while in "reality" one would rather be interested in minimizing the amount of work to be done (i.e. in the *optimization*) when it is ensured that the problem is feasible. We define the cost of a solution to be the number of "loaded" moves (pushes, pulls, lifts); the unloaded motion of the robot is free.

   The only results on optimization of SOKOBAN can be found in [113]. We have taken the basic edge gadget from it. These results were never published and used third dimension to work or considered a slightly modified SOKOBAN problem [34].

2. Several attempts have been made to make the puzzles "more tractable" by limiting the robot's capabilities [41, 38, 40], i.e. by considering BMP(k, p, l) with $l = 0$ and small values of k, p satisfying $kp = 0$. The exact complexity of some of the problems is still unknown; others have been shown PSPACE-complete [23, 33, 66]. The only known "easier" (NP-complete) problem is a somewhat artificial Push-X version (see above), which restricts the robot's paths, rather than its power.

   Our proof of NP-hardness of the optimization problem holds for an arbitrarily powerful robot. We also describe a natural class of *open* BMP instances, for which the optimization problem is in NP.

3. Leakage is a major problem in proving hardness of puzzles with all blocks movable. To constrain the robot's motion certain configurations have to be used: e.g., if the robot can push up to k boxes, a $(k + 1) \times (k + 1)$ square of boxes can be considered fixed to the floor, a wall of thickness more than k can be considered rigid [41, 40], etc. Same configurations

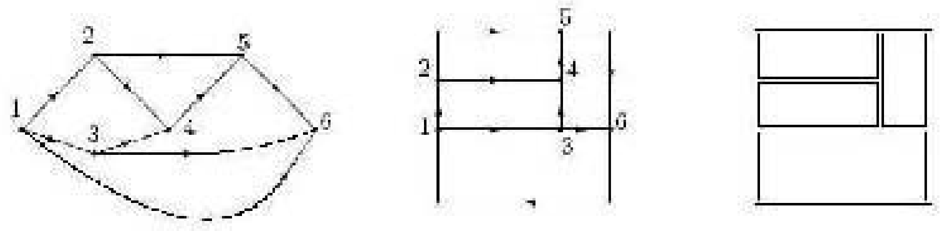


Figure 58: Planar embedding and floor map.

work for constraining of a pull-only robot—once disassembled, these configurations can never be put together. Yet, if the robot can both pull and push (or lift), then no obvious construction (if any at all!) is “heavy” enough to serve as an obstacle for the robot.

In our proof the wall thickness is constant and the proof holds for an arbitrarily powerful robot.

4. In [41] and [40] the authors contrasted their work to “all previous approaches of building circuits based on graphs, which seem to inherently require [problematic] crossings.” In fact, one of the first proofs of NP-hardness of SOKOBAN [45] was based on *Planar* 3-SAT problem and did not use any crossovers. Our construction does not require crossings either, since it is by reduction from HC for *planar* graphs.

The Reduction

The reduction is from the Hamiltonian cycle (HC) problem for planar directed graphs with each node v satisfying $outdegree(v) + indegree(v) = 3$, which is NP-complete by [109]. Let $G = (N, A)$ be such a graph with $|N| = n$. We construct a BMP(1,0,0)-F instance from G such that G contains a HC iff the BMP instance is solvable in $3n - 2$ pushes.

First, embed G in the plane in such a way that the edges of G are drawn with vertical and horizontal segments (Figure 58, left and center). Such an embedding is possible and can be constructed from G in polynomial time [80]. We then use the embedding as a “floor map” for constructing a BMP(1,0,0)-F instance. Each edge of G becomes a corridor of width 1 and every node of G becomes a “T-intersection” of 2 corridors (Figure 52, center and right). Next, we place a *node gadget* (Figure 59, left) in each node of G and an *edge gadget* (Figure 59, right) in the middle of every edge to emulate the direction

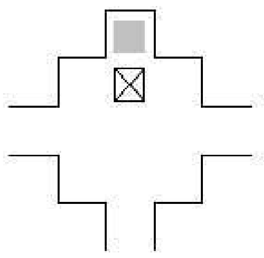
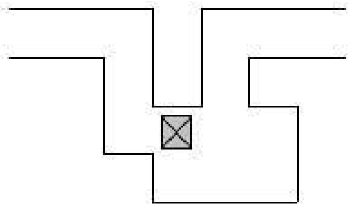



Figure 59: Node (left) and edge (right) gadgets. Boxes (in their initial positions) are marked with ⊠, boxes' target positions are marked with light grey

of the edge. (We may need to lengthen the corridors to have enough space for inserting the edge gadgets.) Note that in the edge gadget, the initial and the target positions of the box coincide. It is easy to see that the edge gadget is passable in one direction only (with 2 pushes per each pass). The robot is initially placed inside a corridor. The robot's goal will be to put the boxes in the target positions.

If G has an HC, then the constructed BMP(1,0,0)-F instance can be solved in $3n - 2$ pushes. Indeed, the robot will follow the HC in G, traveling along $n{-}1$ edges (spending 2 pushes per edge) and pushing all the boxes in the node gadgets in the corresponding target positions (1 push per node). If G is not Hamiltonian, then in order to visit all the nodes, the robot needs to travel twice along at least one edge of G, so the total number of pushes will be not less than $n \cdot 2 + n = 3n$. Thus, the constructed BMP(1,0,0)-F instance can be solved with $3n - 2$ pushes iff G is Hamiltonian.

Results

From the preceding discussion follows

Lemma 7.14. BMP$(1, 0, 0)$-F *is NP-hard.*

We shall now strengthen this result in several aspects. First, observe that if the robot is only allowed to *pull* 1 box (BMP$(0, 1, 0)$-F), the same edge gadget can be used to model the direction on an edge – the only difference is that the direction of the edge is now reversed. If, in addition, the initial and target positions of the boxes in node gadgets are swapped, the same reduction works for BMP$(0, 1, 0)$-F. Hence, Corollary *BMP*$(0, 1, 0)$*-F* is NP-hard. When the robot is allowed to *lift* a box, the directionality of the corridors (edges) is lost (the robot can travel in both directions). In fact, the edge gadget may now be simplified to just be a corridor with a box in target position. Still, the cost of traveling through an edge is 2. The BMP instance now models a planar undirected cubic graph. The HC problem for planar undirected 3-connected cubic graphs having no face with fewer than 5 edges is NP-complete [59]. Thus, our reduction is valid for BMP$(0, 0, 1)$-F as well: Corollary *BMP*$(0, 0, 1)$-*F* is NP-hard. Secondly, observe that if G is Hamiltonian, the path of the robot in the proposed solution is non-self-intersecting. Thus, Corollary *BMP*$(1, 0, 0)$-*F*-*X*, *BMP*$(0, 1, 0)$-*F*-*X* and *BMP*$(0, 0, 1)$-*F*-*X* are NP-hard. Next, observe that we could have assigned numbers to the boxes and target locations. The boxes and the target locations in node gadgets could have been labeled 1 through n and the boxes in the edge gadgets (they are already in target positions) – $n + 1$ to $2n$. The reduction above would not change and thus Corollary #*BMP*$(1, 0, 0)$-*F*, #*BMP*$(0, 1, 0)$-*F* and #*BMP*$(0, 0, 1)$-*F* are NP-hard. Giving the robot the power to push, pull or lift an arbitrary number of boxes would not change the reduction (essentially, the edge gadget is just an "energy waster"). So, Corollary *BMP*(k, p, l)-*F* is NP-hard for any $(k, p, l) \neq (0, 0, 0)$. Since all of the above observations work independently of each other, Corollary [#]*BMP*(k, p, l)-*F[-X]* is NP-hard for any $(k, p, l) \neq (0, 0, 0)$. Finally, we can replace the rigid walls of the corridors by walls of boxes of thickness 2 and change the gadgets as shown in Figure 60. The reduction will still be in place. Indeed, even if the robot has enough power to break through a wall, it would not benefit from doing so, since it would still need to spend too much of a workload before getting to a node gadget. Thus, we have the main result:

Theorem 7.15. *All variations of BMP are NP-hard, i.e.* [#]BMP(k, p, l)[-F][-X] *is NP-hard for any* $(k, p, l) \neq (0, 0, 0)$*, including infinite values of* k, p, l.

As mentioned above, if G is Hamiltonian, the corresponding BMP(1,0,0)-F instance can be solved in $3n - 2$ pushes, while if G is non-Hamiltonian, the number of pushes, needed to solve the instance, is at least $3n$. This shows that (unless P=NP) there exist no Fully Polynomial Time Approximation Scheme (FPTAS) for the problem. Indeed, suppose, that there exists an algorithm, which, for any $\epsilon > 0$, finds a solution, requiring at most $1 + \epsilon$ times the optimum pushes; and that such an algorithm runs in time, polynomial in $1/\epsilon$. Take $\epsilon < \frac{2}{3n-2}$. Then, the algorithm would output a solution of cost less than $3n$ iff G is Hamiltonian. Since this argument works for all versions of the problem, we have Corollary Unless P=NP, there exists no FPTAS for [#]*BMP*(k, p, l)*[-F][-X]*.

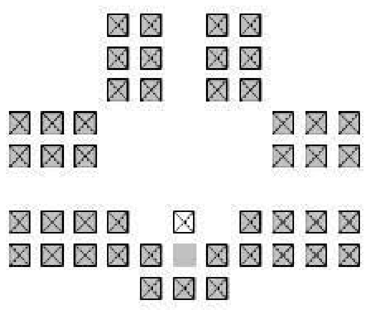
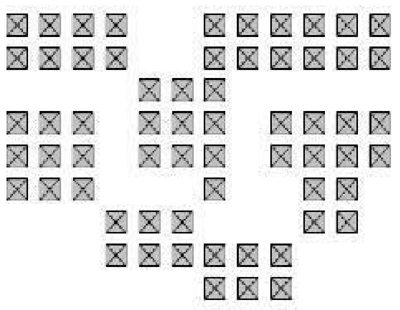



Figure 60: Node (left) and edge (right) gadgets with all blocks movable.

A Realistic Assumption

To prove hardness of BMP, we have constructed some gadgets constraining the agent's motion. Moreover, to avoid leakage, the whole instance was "closed" – once inside the warehouse, the agent is constrained to stay there forever, never to be able to come out and report a solution to an NP-hard problem! We define the *Open Box Mover Problem* (OBMP) as BMP restricted to the instances in which the agent can escape to infinity from the initial position. OBMP retains all the notation introduced in BMP: # , (k, p, l), -F, -X.

Lemma 7.16. *OBMP is NP-hard for all formulations for which BMP is NP-hard.*

Proof. In the constructions used for proving hardness of BMP we could initially put the agent in the edge, adjacent to the unbounded face of the graph, and make a hole in the wall close to the agent's initial position. Having the ability to escape to infinity does not change the cost of a feasible solution (since we only count the workload, not the total travel of the agent). Thus, the reduction, which worked for a BMP formulation, also works for the corresponding version of OBMP. □

Although openness has no impact on optimality, it has drastic effect on feasibility: every instance of OBMP(k, p, l) with $l > 0$ is feasible. Indeed, if the agent can lift and carry the boxes ($l > 0$), he can go to a far point ("infinity"), return to a box, carry it to infinity, return to another box, carry it to infinity and so on. Now, that he has all the boxes at infinity, he can start bringing the boxes back one by one to their target positions. If there are N pixels in the floor map, there are no more than N boxes in the instance. So, the point at the distance of $2N$ from the exit from the warehouse is far enough to be the "infinity" point, to which the agent can carry all the boxes one by one. Thus, any instance of OBMP(k, p, l) with $l > 0$ is feasible. Moreover, it is solvable in at most $O(N^2)$ moves and therefore is in NP.

Theorem 7.17. [#]OBMP(k, p, l)[-F][-X] *with* $l>0$ *is NP-complete.*

7.4 The Snowblower Problem

In this section we introduce the *snowblower problem* (*SBP*), a new optimization problem that is closely related to milling problems and to some material-handling problems. The objective in the SBP is to compute a short tour for the snowblower to follow to remove all the snow from a domain (driveway, sidewalk, etc.). When a snowblower passes over each region along the tour, it displaces snow into a nearby region. The constraint is that if the snow is piled too high, then the snowblower cannot clear the pile.

We give an algorithmic study of the SBP. We show that in general, the problem is NP-complete, and we present polynomial-time approximation algorithms for removing snow under various assumptions about the operation of the snowblower. Most commercially available snowblowers allow the user to control the direction in which the snow is thrown. We differentiate between the cases in which the snow can be thrown in any direction, in any direction except backwards, and only to the right. For all cases, we give constant-factor approximation algorithms; the constants increase as the throw direction becomes more restricted. We focus on the integral orthohedral version of the problem, in which the boundary edges of the domain are parallel to the coordinate axes and the coordinates of its vertices are integral.

Our results are also applicable to robotic vacuuming (or lawnmowing) with bounded capacity dust bin and to some versions of material-handling problems, in which the goal is to rearrange cartons on the floor of a warehouse.

A snowblower is a "material shifting machine," which lifts snow and deposits it nearby. The goal is to dispose of all the snow, moving it outside the driveway. There is a skill in making sure that the deposited piles of snow do not grow higher than the maximum depth capacity of the snowblower. This crystallizes into an algorithmic question, which we have called the *Snowblower Problem* (*SBP*): How does one optimally use a snowblower to clear a given polygonal region?

The SBP shows up in other contexts: Consider a mobile robot that is equipped with a device that allows it to pick up a carton and then place the carton down again in a location just next to it, possibly on a stack of cartons. With each such operation, the robot shifts a unit of "material". The SBP models the problem in which the robot is to move a set of boxes to a specified destination in the most efficient manner, subject to the constraint that it cannot stack boxes higher than a capacity bound.

In a third motivating application, consider a robotic lawnmower or vacuum cleaner that has a catch basin for the clippings, leaves, dust, or other debris. The goal is to remove the debris from a region, with the constraint that the catch basin must be emptied (e.g., in the compost pile) whenever it gets full.

The SBP is related to other problems on milling, vehicle routing, and traveling salesman tours, but there are two important new features: (a) material must be moved (snow must be thrown), and (b) material may not pile up too high.

While the SBP arises naturally in these other application domains, we use the terminology of snow removal.

The objective of the SBP is to find the shortest snowblower tour that clears a domain P, assumed to be initially covered with snow at uniform depth 1. An important parameter of the problem is the maximum snow depth $D > 1$ through which the snowblower can move. At all times no point of P should have snow of greater depth than D. The snow is to be moved to points outside of P. We assume that each point outside P is able to receive arbitrarily much snow (i.e., that the driveway is surrounded by a "cliff" over which we can toss as much snow as we want).[13]

Snowblowers offer the user the ability to control the direction in which the snow is thrown. Some throw directions are preferable over others; e.g., throwing the snow back into the user's face is undesirable. However, it can be cumbersome to change the throw direction too frequently during the course of clearing. Thus, we consider three *throw models.* In the *default* model throwing the snow backwards is allowed. In the *adjustable-throw* model the snow can be thrown only to the left, right, or forward. In the *fixed-throw* model the snow is always thrown to the right. Even though it seems silly to allow the throw direction to be back into one's face, the default model is the starting point for the analysis of other models; it is also equivalent to the problem of *vacuum-cleaning* a floor.

[13]The "cliff" assumption accurately models the capacitated-vacuum-cleaner problem for which there is a (central) "dustpan vac" in the baseboard, where a robotic vacuum cleaner may empty its load [1] and applies also to urban snow removal using snow melters [2] or disposing off the snow into a river.

Results We show that the SBP is NP-complete for multiply connected domains. Our main results are constant-factor approximation algorithms for each of the three throw models, assuming $D \geq 2$. The approximation ratio of our algorithms increases as the throw direction becomes more restricted.

Default model, Theorem. 7.23		
D	2 or 3	any $D \geq 4$
Approximation	6	8

Adjustable throw, Theorem. 7.26	
D	any $D \geq 2$
Approximation	$4 + 3D/\lfloor D/2 \rfloor$

Fixed throw, Theorem. 7.29	
D	any $D \geq 2$
Approximation	$34 + 24D/\lfloor D/2 \rfloor$

The SBP is closely related to milling and lawn-mowing problems, which have been studied extensively in the NC-machining and computational-geometry literatures; see e.g., [11, 10, 67]. The SBP is also closely related to material-handling problems, in which the goal is to rearrange a set of objects (e.g., cartons) within a storage facility; see [41, 33] and Section 7.3 of this thesis. The SBP may be considered as an intermediate point between the TSP/ lawnmowing/milling problems and material-handling problems. Indeed, for $D = \infty$, the SBP is that of optimal milling. Unlike most material-handling problems, the SBP formulation allows the material (snow) to pile up on a single pixel of the domain, and it is this compressibility of the material that distinguishes the SBP from previously studied material-handling problems. With TSP and related problems, every pixel is visited only a constant number of times, whereas with material-handling problems, pixels may have to be visited a number of times exponential in the input size. For this reason, material-handling problems are not even known to be in NP [41, 33], in contrast with the SBP. Note that in material handling problems the objective is to minimize *workload* (distance traveled while loaded), while in the SBP (as in the milling/mowing problems) the objective is to minimize total travel distance (loaded or not).

The SBP is also related to the earth-mover's distance (EMD), which is the minimum amount of work needed to rearrange one distribution (of earth, snow, etc.) to another; see [31]. In the EMD literature, the question is explored mostly from an existential point of view, rather than planning the actual process of rearrangement. In the SBP, we are interested in optimizing the length of the tour, and we do not necessarily know in advance the final distribution of the snow after it has been removed from P.

Notation The input is a polygonal domain, P. Since we are mainly concerned with proving constant factor approximation algorithms, it suffices to consider distances measured according to the L_1 metric. We consider the snowblower to be an (axis-parallel) unit square that moves horizontally or vertically by unit steps. This justifies our assumption, in most of our discussion, that P is an integral-orthogonal simple polygon, which is comprised of a union of *pixels* – (closed) unit squares with disjoint interiors and integral coordinates.

We say that two pixels are *adjacent* or *neighbors* if they share a side; the *degree* of a pixel is the number of its neighbors. For a region $R \subseteq P$ (subset of pixels), let G_R denote the *dual graph* of R, having a vertex in the center of each pixel of R and edges between adjacent pixels.

A pixel of degree less than four is a *boundary pixel.* For a boundary pixel, a side that is also on the boundary of P is called a *boundary side.* The set of boundary sides, ∂P, forms the boundary of P. We assume that the elements of ∂P are ordered as they are encountered when the boundary of P is traversed counterclockwise.

An *articulation vertex* of a graph G is a vertex whose removal disconnects G. We assume that G_P has no articulation vertices. (Our algorithms can be adapted to regions having articulation vertices, at a possible increase in approximation ratio.)

Algorithms Overview Our algorithms proceed by clearing the polygon Voronoi-cell-by-Voronoi-cell, starting from the Voronoi cell of the garage g — the pixel on the boundary of P at which the snowblower tour starts and ends. The order of the boundary sides in ∂P provides a natural order in which to clear the cells. We observe that the Voronoi cell of each boundary side is a tree of one of two special types, which we call *lines* and *combs.* We show how to clear the trees efficiently in each of the throw models. We prove that our algorithms give constant-factor approximations by charging the lengths of the tours produced by the algorithms to two lower bounds, described below.

Voronoi Decomposition For a pixel $p \in P$ let $V(p)$ denote the element of ∂P closest to p. In case of ties, the tie-breaking rule (see below) is applied.

Inspired by computational-geometry terminology, we call $V(p)$ the *Voronoi side* of p. We let $\delta(p)$ denote the length of the path from p to the pixel having $V(p)$ as a side. For a boundary side $e \in \partial P$ we let Voronoi(e) denote the (possibly, empty) set of pixels, having e is the Voronoi side: Voronoi$(e) = \{p \in P | V(p) = e\}$. We call Voronoi$(e)$ the *Voronoi cell* of e. The Voronoi cells of the elements of ∂P form a partition of P, called the *Voronoi decomposition* of P. We remark that our Voronoi decomposition is a discrete version of the Voronoi diagram of the edges of P [14].

A set of pixels $\mathcal{L}$ whose dual graph $G_{\mathcal{L}}$ is a straight path or a path with one bend, is called a *line*. Each line $\mathcal{L}$ has a *root* pixel p, which corresponds to one of the two leaves of $G_{\mathcal{L}}$, and a *base*, $e \in \partial P$, which is a side of p.

A *(horizontal) comb* $\mathcal{C}$ is a union of pixels consisting of a set of vertically adjacent (horizontal) rows of pixels, with all of the rightmost pixels (or all of the leftmost pixels) in a common column. (A vertical comb is defined similarly; however, by our tie breaking rules, we need consider only horizontal combs.) A comb is a special type of *histogram* polygon [30]. The common vertical column of rightmost/leftmost pixels is called the *handle* of comb $\mathcal{C}$, and each of the rows is called a *tooth*. A *leftward* comb has its teeth extending leftwards from the handle; a *rightward* comb is defined similarly. The pixel of a tooth that is furthest from the handle is the *tip* of the tooth. The topmost row is the *wisdom tooth* of the comb. The *root* pixel p of the comb is either the bottommost or topmost pixel of the handle, and its bottom or top side, $e \in \partial P$, is the *base* of the comb. See Figure 61, left. The union of a leftward comb and a rightward comb having a common root pixel is called a *double-sided comb*.

Tie Breaking Our rules for finding $V(p)$ for a pixel p that is equidistant between two or more boundaries is based on the direction of the shortest path from p to $V(p)$; vertical edges are preferred to horizontal, going down has higher priority than going up, going to the right — than going left. In fact, any tie-breaking rule can be applied as long as it is applied consistently. The particular choice of the rule only affects the orientation of the combs.

Voronoi Cell Structure An analysis of the structure of the Voronoi partition under our tie breaking rules gives:

Lemma 7.18. *For a side $e \in \partial P$, the Voronoi cell of e is either a line (whose dual graph is a straight path), or a comb, or a double-sided comb. By our tie-breaking rule, the combs may appear only as the Voronoi cells of horizon-*

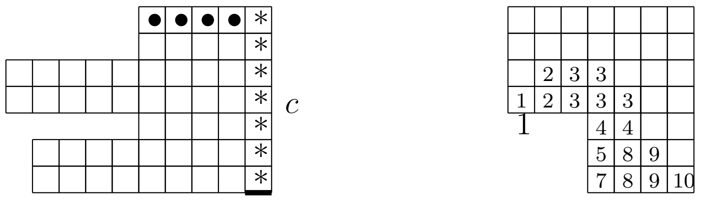


Figure 61: Left: a comb. The base is bold. The pixels in the handle are marked with asterisks, the pixels in the wisdom tooth are marked with bullets. Right: Voronoi cells. The sides of ∂P are numbered $1 \ldots 28$ counterclockwise. The pixels in the Voronoi cell of a side are marked with the corresponding number. Voronoi cell of side 3 is a comb; Voronoi cells of sides 6, 11, 17, 25, 28 are empty; cells of sides 1, 7, 10, 18, 24 are lines, comprised of just one pixel; cells of the other edges are lines with more than one pixel.

tal edges. The double-sided combs may appear only as the Voronoi cells of (horizontal) edges of length 1.

Let p be a boundary pixel of P, let $e \in \partial P$ be the side of p such that $p \in \text{Voronoi}(e)$. We denote Voronoi(e) by $\mathcal{T}(p)$ or $\mathcal{T}(e)$, indicating that it is a unique tree (a line or a comb) that has p as the root and e as the base.

Lower Bounds We exhibit two lower bounds on the cost of an optimal tour, the *snow lower bound*, based on the number of pixels, and the *distance lower bound*, based on the Voronoi decomposition of the domain. At any time let $s(R)$ be the set of pixels of R covered with snow and also, abusing notation, the number of these pixels. Let $d(R) = \frac{1}{D} \sum_{p \in s(R)} \delta(p)$.

Lemma 7.19. *Let R be a subset of P with the snowblower starting from a pixel outside R. Then $s(R)$ and $d(R)$ are lower bounds on the cost to clear R.*

Proof. For the snow lower bound, observe that region R cannot be cleared with fewer than $s(R)$ snowblower moves because each pixel of $s(R)$ needs to be visited.

For the distance lower bound, observe that, in order to clear the snow initially residing on a pixel p, the snowblower has to make at least $\delta(p)$ moves. When the snow from p is carried to the boundary of P and thrown away, the snow from at most $D-1$ other pixels can be thrown away simultaneously. Thus, a region R cannot be cleared with fewer than $d(R)$ moves. $\square$ $\square$

NP-Completeness It is known [79, 105] that the Hamiltonian path problem in cubic grid graphs is NP-complete. The problem can be straightforwardly reduced to SBP. If G is a cubic grid graph, construct an (integral orthohedral) domain P such that $G = G_P$. Since G_P is cubic, each pixel $p \in P$ is a boundary pixel, thus, the snowblower can throw the snow away from p upon entering it. Hence, SBP on P is equivalent to TSP on G, which has optimum less than $n+1$ iff G is Hamiltonian (where n is the number of nodes in G). The reduction works for any $D \geq 1$.

The algorithms proposed in this thesis show that any domain can be cleared using a set of moves of cardinality polynomial in the number of pixels in the domain, assuming $D \geq 2$. Thus, we obtain

Theorem 7.20. *If $D \geq 2$, the SBP is NP-complete, both in the default model and in the adjustable throw model, for inputs that are polygonal domains with holes.*

Approximation Algorithm for the Default Model

In this section we give an 8-approximation algorithm for the case when the snow can be thrown in *all four* directions. We first show how to clear a line efficiently with the operation called *line-clearing*. We then introduce another operation, the *brush*, and show how to clear a comb efficiently with a sequence of line-clearings and brushes. Finally, we splice the subtours through each line and comb into a larger tour, clearing the entire domain. The algorithm for the default model, developed in this section, serves as a basis for the algorithms in the other models.

Clearing a Line Let $\mathcal{L}$ be a line of pixels; let p and e be its root and the base. We are interested in clearing lines for which the base is a boundary side, i.e., $e \in \partial P$. Let $\ell = s(\mathcal{L})$; let the first J pixels of $\mathcal{L}$ counting from p be clear. We assume that p is already clear ($J > 0$); the snow from it was thrown away through the side e as the snowblower first entered pixel p. Let $\mathcal{L}|J$ denote $\mathcal{L}$ with the J pixels clear; let $\ell - J = kD + r$.[14] Denote by $(\mathcal{L}|J)_D$ the first kD

[14]For ease of presentation, we adapt the following convention. For $d \in \{D, \lfloor D/2 \rfloor\}$ and an integer w we understand the equality $w = ad + b$ as follows: b and a are the remainder and the quotient, respectively, of w divided by d.

pixels of $\mathcal{L}|J$ covered with snow; denote by $\mathcal{L}_r$ the last r pixels on $\mathcal{L}|J$. The idea of decomposing $\mathcal{L}|J$ into $(\mathcal{L}|J)_D$ and $\mathcal{L}_r$ is that the snow from $(\mathcal{L}|J)_D$ is thrown away with k "fully-loaded" throws, and the snow from $\mathcal{L}_r$ is thrown away with (at most one) additional "under-loaded" throw.

We clear line $\mathcal{L}$ starting at p by moving all the snow through the base e and returning back to p. The basic clearing operation is a back throw. In a back throw the snowblower, entering a pixel u from pixel v, throws u's snow backward onto v. Starting from p, the snowblower moves along $\mathcal{L}$ away from p until either the snowblower moves through D pixels covered with snow or the snowblower reaches the other end of $\mathcal{L}$; this is called the *forward pass*. Next, the snowblower makes a U-turn and moves back to p, pushing all the snow in front of it and over e; this is called the *backward pass*. A forward and backward pass that clears exactly D units of snow is called a *D-full pass*.

Lemma 7.21. *For arbitrary $D \geq 4$ the line-clearing cost is at most $2s(\mathcal{L} \setminus p) + 4d(\mathcal{L}|J)$. For $D = 2, 3$ the line-clearing cost is at most $2s(\mathcal{L} \setminus p) + 2d(\mathcal{L}|J)$. If every pass is D-full, the cost is $4d(\mathcal{L}|J)$ for $D \geq 4$ and $2d(\mathcal{L}|J)$ for $D = 2, 3$.*

Proof. The clearing cost is $c(\mathcal{L}|J) = c((\mathcal{L}|J)_D) + c(\mathcal{L}_r) = \sum_{i=1}^{k} 2(J - 1 + iD) + 2(\ell - 1) = 2kJ + Dk(k+1) - 2k + 2(\ell - 1)$. The *snow* lower bound of $\mathcal{L} \setminus p$ is $s(\mathcal{L} \setminus p) = \ell - 1$. The *distance* lower bound of $(\mathcal{L}|J)_D$ is $d((\mathcal{L}|J)_D) = \frac{1}{D}\sum_{i=1}^{kD}(J + i) = kJ + k(kD+1)/2$.

Thus,

$$c(\mathcal{L}|J) = 2s(\mathcal{L} \setminus p) + \left(2 + \frac{D-3}{J + (Dk+1)/2}\right) d((\mathcal{L}|J)_D)$$

If every pass is a D-full pass, then $c(\mathcal{L}_r) = 0$. Therefore, $c(\mathcal{L}|J) = c((\mathcal{L}|J)_D) = \left(2 + \frac{D-3}{J+(Dk+1)/2}\right) d((\mathcal{L}|J)_D)$. □ □

Clearing a Comb Let $\mathcal{C}$ be a comb with the root p, base e, and handle $\mathcal{H}$ of length H. Let $\ell_1 \ldots \ell_H$ be the lengths of the teeth of the comb. Since we are interested in clearing combs for which the base e is a boundary side ($e \in \partial P$), we assume that pixel p is already clear — the snow from it was thrown away through e as the snowblower first entered p.

Our strategy for clearing $\mathcal{C}$ is as follows. While there exists a line $\mathcal{L} \subset \mathcal{C}$ rooted at p, such that $s(\mathcal{L}) \geq D$, we perform as many D-full passes on $\mathcal{L}$ as we can. When no such $\mathcal{L}$ remains, we call the comb *brush-ready* and we use another clearing operation, the *brush*, to finish the clearing.

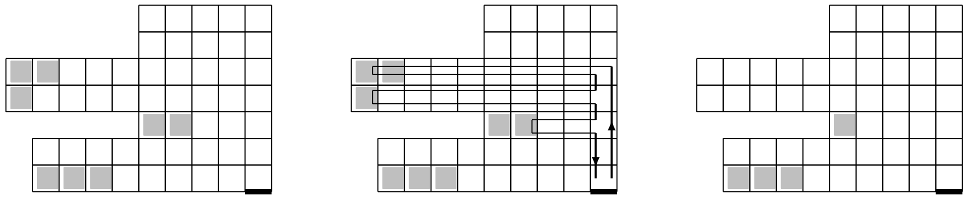


Figure 62: Left: a brush-ready comb. The snow is shown in light gray. Center: a brush, $D = 4$; the part of the brush, traveling through the handle, is bold. Right: the comb after the brush.

A brush, essentially, is a "capacitated" depth-first-search. Among the teeth of a brush-ready comb that are not fully cleared, let t be the tooth, furthest from the base. In a brush, we move the snowblower from p through the handle, turn into t, reach its tip, U-turn, come back to the handle (pushing the pile of snow), turn onto the handle, move by the handle back towards p until we reach the next not fully cleared tooth, turn onto the tooth, and so on. We continue clearing the teeth one-by-one in this manner until D units of snow have been moved (or all the snow on the comb has been moved). Then we push the snow to p through the handle and across e. This tour is called a *brush* (Figure 62).

Lemma 7.22. *For arbitrary $D \geq 4$ the comb $\mathcal{C}$ can be cleared at a cost of at most $4s(\mathcal{C} \setminus p) + 4d(\mathcal{C} \setminus p)$ (at most $4s(\mathcal{C} \setminus p) + 2d(\mathcal{C} \setminus p)$ for $D = 2, 3$).*

Proof. If $s(\mathcal{C} \setminus p) < D$, then the cost of clearing is just $2s(\mathcal{C} \setminus p)$, so suppose, $s(\mathcal{C} \setminus p) \geq D$. Let B be the number of brushes used; let $\mathcal{B}$ be the set of pixels cleared by the brushes. For $b = 1 \dots B$ let t_b and t'_b be the first and the last tooth visited during the bth brush. For $b \in \{1 \dots B-1\}$ the bth brush enters at least 2 teeth, so $t_b > t'_b \geq t_{b+1}$.

Each brush can be decomposed into two parts: the part traveling through the teeth and the part traveling through the handle (Figure 62). Since each tooth is visited during at most 2 brushes, the length of the first part is at most 4 times the size of all teeth, that is, $4s(\mathcal{C} \setminus \mathcal{H})$. The total length of the second part of all brushes is $2\sum_{b=1}^{B}(t_b - 1)$. Thus, the cost of the "brushing" is

$$c(\mathcal{B}) \leq 2\sum_{b=1}^{B}(t_b - 1) + 4s(\mathcal{C} \setminus \mathcal{H}) \leq 2\sum_{b=2}^{B} t_b + 4s(\mathcal{C} \setminus p) - 2 \qquad (5)$$

since $t_1 \leq H$ and $H \geq 2$ (for otherwise $\mathcal{C}$ is a line).

There are exactly D pixels cleared during each brush $b \in \{0 \dots B-1\}$, and each of these pixels is at distance at least $t_{b'}$ from the base of the comb. Thus, the *distance* lower bound of the pixels, cleared during brush b, is at least $t_{b'}$. Consequently, the *distance* lower bound of $\mathcal{B}$

$$d(\mathcal{B}) \geq \sum_{b=1}^{B} t_{b'} \geq \sum_{b=1}^{B-1} t_{b+1} = \sum_{b=2}^{B} t_b \tag{6}$$

From (5) and (6), $\mathcal{B}$ can be cleared at a cost of at most $2d(\mathcal{B}) + 4s(\mathcal{C} \setminus p)$.

Let $\mathcal{P} \subseteq \mathcal{C}$ be the pixels, cleared during the line-clearings. By our strategy, during each line-clearing, every pass is D-full; thus, by Lemma 7.21, $\mathcal{P}$ can be cleared at a cost of at most $4d(\mathcal{P})$ (or $2d(\mathcal{P})$ if $D = 2, 3$). Since $\mathcal{P}$ and $\mathcal{B}$ are snow-disjoint and $\mathcal{P} \cup \mathcal{B} = \mathcal{C} \setminus p$, the lemma follows. □

The above analysis is also valid in the case when the handle is initially clear. This is the case when the second side of a double-sided comb is being cleared. Thus, a double-sided comb can be cleared within the same bounds on the cost of clearing.

Clearing the Domain Now that we have defined the operations which allow us to clear efficiently lines and combs, we are ready to present the algorithm for clearing the domain.

Theorem 7.23. *For arbitrary $D \geq 4$ (resp., $D = 2, 3$) an 8-approximate (resp., 6-approximate) tour can be found in polynomial time.*

Proof. Let $p_1, \dots, p_M$ be the boundary pixels of P as they are encountered when going around the boundary of P counterclockwise starting from $g = p_1$; let $e_1, \dots, e_M \in \partial P$ be the boundary sides of $p_1, \dots, p_M$ such that $e_i = Ve(p_i)$, $i = 1 \dots M$. The polygon P can be decomposed into disjoint trees $\mathcal{T}(p_1), \dots, \mathcal{T}(p_M) = \mathcal{T}(e_1), \dots, \mathcal{T}(e_M)$ with the bases $e_1 \dots, e_M$, where each tree $\mathcal{T}(e_i)$ is either a line or a comb.

Our algorithm clears P tree-by-tree starting with $\mathcal{T}(e_1) = \mathcal{T}(g)$. By Lemmas 7.21 and 7.22, for $i = 1 \dots M$, the tree $\mathcal{T}(p_i) \setminus p_i$ can be cleared at a cost of at most $4s(\mathcal{T}(p_i) \setminus p_i) + 4d(\mathcal{T}(p_i) \setminus p_i)$ starting from p_i and returning to p_i. Since $\bigcup_1^M \mathcal{T}(p_i) \setminus p_i = P \setminus \{p_1 \dots p_M\}$, the interior of P can be cleared at a cost of at most $c(P \setminus \{p_1 \dots p_M\}) = 4s(P \setminus \{p_1 \dots p_M\}) + 4d(P \setminus \{p_1 \dots p_M\}) \leq 4s(P \setminus g) + 4d(P \setminus g) - 4M + 4$.

Finally, the tours clearing the interior of P can be spliced into a tour, clearing P at a cost of at most $2M$. Since the optimum is at least $s(P \setminus g)$ and is at least $d(P \setminus g)$, the theorem follows. □

Other Models

In this section we give approximation algorithms for the case when the throw direction is restricted. Specifically, we first consider the adjustable-throw-direction formulation. This is a convenient case for the snowblower operator who does not want the snow thrown in his face. We then consider the fixed-throw-direction formulation, which assumes that the snow is always thrown to the right.

We remark that the relatively low approximation factors of the algorithms for the default model, presented in the previous section, were due to a very conservative clearing: the snow from *every* pixel $p \in P$ was thrown through the Voronoi side $V(p)$. Unfortunately, it seems hard to preserve this appealing property if throwing back is forbidden. The reason is that the comb in the Voronoi cell Voronoi(e) of a boundary side $e \in \partial P$ often has a "staircase"-shaped boundary; clearing the first "stair" in the staircase cannot be done without throwing the snow onto a pixel of Voronoi(e'), where $e' \neq e$ is another boundary side. This is why the approximation factors of the algorithms in this section are higher than those in the previous one.

Adjustable Throw Direction

In the adjustable-throw model the snow cannot be thrown backward but can be thrown in the three other directions. To give a constant-factor approximation algorithm for this case, we show how to emulate line-clearings and brushes avoiding back throws (Figure 63). The approximation ratios increase slightly in comparison with the default model.

Line-clearing We can emulate a (half of a) pass by a sequence of moves, each with throwing the snow to the left, forward or to the right (Figure 63, left and center). Thus, the line-clearing may be executed in the same way as it was done if the back throws were allowed. The only difference is that now the snow is moved to the base when the snow from only $\lfloor D/2 \rfloor$ pixels (as opposed to D pixels) of the line is gathered.

Lemma 7.24. *The line-clearing cost is at most* $3D/\lfloor D/2 \rfloor d(\mathcal{L}|J) + 2s(\mathcal{L} \setminus p)$. *If every pass is* $\lfloor D/2 \rfloor$*-full, the cost is* $3D/\lfloor D/2 \rfloor d(\mathcal{L}|J)$.

Proof. Let $\ell - J = k'\lfloor D/2 \rfloor + r'$. Let $(\mathcal{L}|J)_{\lfloor D/2 \rfloor}$ be the first $k'\lfloor D/2 \rfloor$ pixels of $\mathcal{L}|J$, let $\mathcal{L}_{r'}$ be its last r' pixels. Then the cost of the clearing of $\mathcal{L}|J$ is $c(\mathcal{L}|J) = c((\mathcal{L}|J)_{\lfloor D/2 \rfloor}) + c(\mathcal{L}_{r'}) = \sum_{i=1}^{k'} 2(J + i\lfloor D/2 \rfloor) + 2\ell = 2k'J + \lfloor D/2 \rfloor k'(k'+1) + 2\ell$.

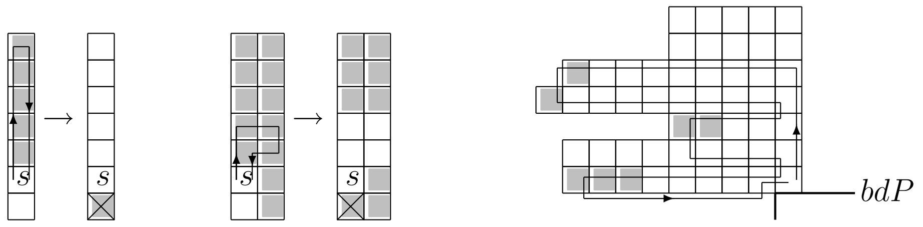


Figure 63: Emulating line-clearing and brush. The (possible) snow locations are in light gray; s is the snowblower. Left: forward and backward passes in the default model; there are D units of snow on the checked pixel. Center: the passes emulation; there is (at most) $2\lfloor D/2 \rfloor$ units of snow on the checked pixel. Right: the snow to be cleared during a brush is in light gray; there are $\lfloor D/2 \rfloor$ light gray pixels.

The lower bounds are given by $s(\mathcal{L} \setminus p) = \ell - 1$ and

$$d((\mathcal{L}|J)_{\lfloor \frac{D}{2} \rfloor}) = \frac{1}{D} \sum_{i=1}^{k' \lfloor \frac{D}{2} \rfloor} (J+i) = \frac{\lfloor \frac{D}{2} \rfloor}{D} \left[k'J + \frac{k'(k' \lfloor \frac{D}{2} \rfloor + 1)}{2} \right] \tag{7}$$

Thus,

$$c(\mathcal{L}|J) \le \frac{D}{\lfloor \frac{D}{2} \rfloor} \left(2 + \frac{2 + \lfloor D/2 \rfloor k' - k'}{k'J + \frac{\lfloor D/2 \rfloor}{2} k'^2 + \frac{k'}{2}} \right) d(\mathcal{L} \setminus p) + 2s(\mathcal{L} \setminus p) \qquad \square$$

□

Brush Brush also does not change too much from the default case. The difference is the same as with the line-clearing: now, instead of clearing D pixels with a brush, we prepare to clear only $\lfloor D/2 \rfloor$ pixels (Figure 63, right). Consequently, the definition of a brush-ready comb is changed — now we require that there is less than $\lfloor D/2 \rfloor$ pixels covered with snow on each tooth of such a comb. Observe that together with each unit of snow, the snow from at most 1 other pixel is moved — thus (although the brush may go outside the comb, as, e.g., in Figure 63), the brush is feasible.

Lemma 7.25. *A comb can be cleared at a cost of* $3D/\lfloor D/2 \rfloor d(\mathcal{C} \setminus p) + 4s(\mathcal{C} \setminus p)$.

Proof. In comparison with the default model (Lemma 7.22) several observations are in place. The number of brushes may go up; we still denote it by B.

The cost of the brushes $1 \dots B-1$ does not change. If the Bth brush has to enter the first tooth, there may be 2 more moves needed to return to the root of the comb (see Figure 63, right); hence, the total cost of the brushing (5) may go up by 2. The *distance* lower bound (6) goes down by $D/\lfloor D/2 \rfloor$. The rest of the proof is identical to the proof of Lemma 7.22 (with Lemma 7.24 used in place of Lemma 7.21). □

Observe that in fact the snow can be removed from *more than* $\lfloor D/2 \rfloor$ pixels during a brush; we just ignore it for now in our analysis. Note that a double-sided comb can also be cleared in the described way.
Clearing the Domain As in the default case (Theorem 7.23),

Theorem 7.26. *A* $(4+3D/\lfloor D/2 \rfloor)$*-approximate tour can be found in polynomial time.*

Comment on the Parity of D We remark that if D is even, the cost of the clearing is the same as it would be if the snowblower were able to move through snow of depth $D+1$ (the slight increase of $6/(D-1)$ in the approximation factor would be due to the decrease of the *distance* lower bound).

Fixed Throw Direction

In reality, changing the throw direction requires some effort. In particular, a snow *plow* does not change the direction of snow displacement at all. In this section we consider the fixed throw direction model, i.e., the case of the snowblower which can only throw the snow to the right. We exploit the same idea as in the previous subsection — reducing the problem in the fixed throw direction model to the problem in the default model. All we need is to show how to emulate line-clearing and brush.

In what follows we retain the notation from the previous subsection.

Lemma 7.27. *The line-clearing cost is at most* $24D/\lfloor D/2 \rfloor d(\mathcal{L}|J)+25s(\mathcal{L}\backslash p)$*. If every pass is* $\lfloor D/2 \rfloor$*-full, the cost is* $24D/\lfloor D/2 \rfloor d(\mathcal{L}|J)$*.*

Proof. We first consider clearing a line whose dual graph is embedded as a single straight line segment and whose base is perpendicular to the segment; we describe the line-clearing, assuming that the line is vertical. Next, we extend the solution to the case when the base is parallel to the edges of the dual graph; this can only be a horizontal line — the first tooth in a (double-)comb. Finally, we consider clearing an L-shaped line; this can only by a tooth together with the (part of the) handle.

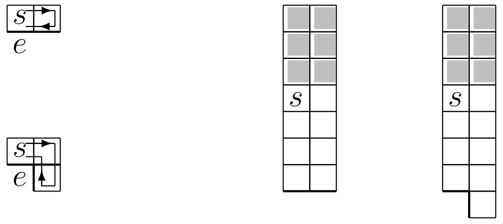


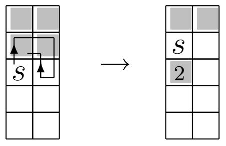
Figure 64: The boundary sides are bold. Left: the double-base setup. e is the base. Right: before the forward pass the snow below the snowblower is cleared on both lines.

A Line $\mathcal{L}$ with $G_{\mathcal{L}} \perp e$. As in the adjustable-throw case (see Figure 63, left and center), to clear $\mathcal{L}$ we will need to use the pixels to the right of $\mathcal{L}$ to throw the snow onto. Let p' be the boundary pixel, following p counter-clockwise around the boundary of P. Before the line-clearing is begun, it will be convenient to have p' clear. Thus, the first thing we do upon entering $\mathcal{L}$ (through p) is clearing p'. Together with returning the snowblower to p it takes 2 or 4 moves (Figure 64, left); we call these moves *the double-base setup.*

Then, the following invariant is maintained during line-clearing. If the snowblower is at a pixel $q \in \mathcal{L}$ before starting the forward pass, all pixels on $\mathcal{L}$ from p to q are clear, along with the pixels to the right of them (Figure 64, right). The invariant holds in the beginning of the line-clearing and our line-clearing strategy respects it.

Each back throw is emulated with 5 moves (Figure 65, left). After moving up by $\lfloor D/2 \rfloor$ pixels (and thus, gathering $2\lfloor D/2 \rfloor$ units of snow on these $\lfloor D/2 \rfloor$ pixels), the snowblower U-turns and moves towards p "pushing" the snow in front of it; a push is emulated with 11 moves (Figure 66).

The above observations already show that the cost of line-clearing increases only by a multiplicative constant in comparison with the adjustable-throw case. A more careful look at the Figures 65 and 66 reveals that: (1) in the push emulation the first two moves are the opposites of the last two, thus, all 4 moves may be omitted – consequently, a push may be emulated by a sequence of only 7 moves; (2) if the boundary side, following e, is vertical, the last push, throwing the snow away from P, may require 9 moves (Figure 65, right); and, (3) when emulating the last back throw in a forward pass, the last 2 of the 5 moves (the move up and the move to the right in Figure 65, left) can be omitted – indeed, during the push emulation, the snowblower may as well start to the right of the snow (see Figure 66). Thus, a line $\mathcal{L}|J$ can be cleared at a cost of



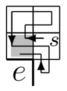


Figure 65: Left: emulating back throw. Right: pushing the $2\lfloor D/2\rfloor$ units of snow away from P and returning the snowblower to p may require 9 moves.

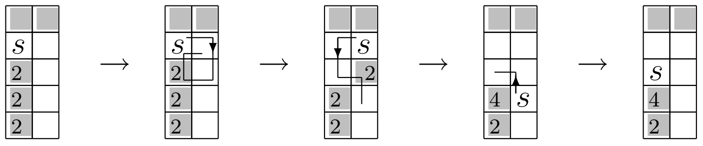


Figure 66: Emulating pushing the snow in front of the snowblower.

$c(\mathcal{L}|J) \leq 4 \quad +\sum_{i=1}^{k'} (J-1+(i-1)\lfloor D/2\rfloor + 5\lfloor D/2\rfloor + 7(J+i\lfloor D/2\rfloor - 1))+ J+5r'+7(\ell-1)$.

A Line $\mathcal{L}$ with $G_{\mathcal{L}}||e$. Consider a horizontal line, extending to the *left* of the base; such a line may represent the first tooth of a comb. The double-base can be cleared with 8 or 12 moves (see Figure 67), the root can be cleared with 3 moves (see Figure 68, left) instead of 9 moves (see Figure 65, right); the rest of the clearing does not change.

Consider now a horizontal line extending to the *right* of the base; such a line may appear as the first tooth in a double-sided comb. The double-base for such a line can be cleared with 3 moves; the rest of the clearing is the same as for the vertical line.

***L*-shaped Line.** An L-shaped line $\mathcal{L}$ consists of a vertical and a horizontal segment. Each of the segments can be cleared as described above.

Thus, *any* line $\mathcal{L}|J$ can be cleared at a cost of at most $c(\mathcal{L}|J) \leq 12 + \sum_{i=1}^{k'} (J-1+(i-1)\lfloor D/2\rfloor + 5\lfloor D/2\rfloor + 7(J+i\lfloor D/2\rfloor - 1))+J+5r'+7(\ell-1)$.

Since the *snow* and *distance* (7) lower bounds do not change, the lemma follows. □

Lemma 7.28. *A comb can be cleared at a cost of* $34s(\mathcal{C}\setminus p)+\frac{24D}{\lfloor D/2\rfloor}d(\mathcal{C}\setminus p)$.

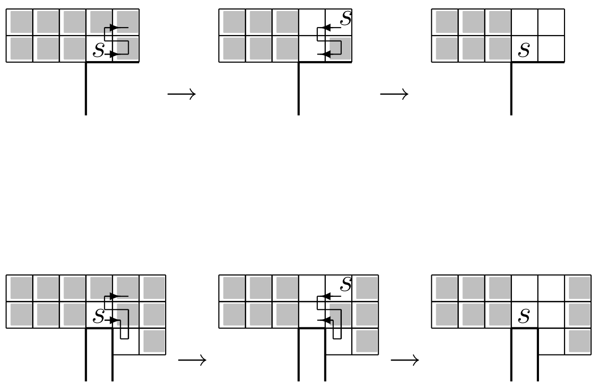


Figure 67: Setting up the double-base for clearing a horizontal line extending to the left of its base. Depending on the direction of the edge adjacent to the base from the right, there are 8 (above) or 12 (below) moves necessary.

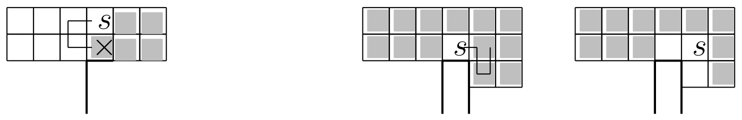


Figure 68: Left: clearing the root of a horizontal, extending to the left, line with 3 moves. There is $2\lfloor D/2 \rfloor$ units of snow on the checked pixel. Right: Setting up the double-base for clearing a horizontal line extending to the right of its base.

Proof. Brush in the fixed throw direction model can be described easily using analogy with: a) brush in default and adjustable-throw models and b) line-clearing in fixed-throw model. As in the adjustable-throw model, we prepare to clear $\lfloor D/2 \rfloor$ pixels during each brush. Same as with line-clearing, we setup the double-base for the comb with at most 12 moves; also, 9 moves per brush may be needed to push the snow away from P through the base. Back throw and push can be emulated with 5 and 7 moves (Figure 65, left and Figure 66). Thus, if the cost of a brush (5) in the default model was, say, c, the cost of the brush in the fixed-throw model is at most $7c + 9$. Since any brush starts with the double-base setup, $c \geq 6$; this, in turn, implies $7c + 9 \leq (51/6)c$. Hence, the cost of clearing $\mathcal{B}$ increases by at most a factor of 51/6.

By Lemma 7.27, the cost of clearing $\mathcal{P}$, $c(\mathcal{P}) \leq 24D/\lfloor D/2 \rfloor d(\mathcal{P})$. The *snow* and *distance* lower bounds do not change in comparison with the adjustable-throw case. The lemma now follows from simple arithmetic. ☐

As in the default and adjustable-throw models (Theorems 7.23, 7.26),

Theorem 7.29. *A* $(34 + \frac{24D}{\lfloor D/2 \rfloor})$*-approximate tour can be found in polynomial time.*

7.5 Hamiltonian Cycles in Triangular Grids

In the previous sections we considered vehicle routing problems in "pixelated" domains, or, equivalently, on square "grid" graphs. In general, grids have proved to be extremely useful in all areas of computer science. Their main usage is as the discrete approximation to a continuous domain or surface. Numerous algorithms in computer graphics, numerical analysis, computational geometry, robotics and other fields are based on grid computations.

Formally, a *square grid*, or *square grid graph* G is induced by a finite subset $\mathbb{G}$ of the infinite integer grid $\mathbb{Z}^2$: the vertices of G are the points in $\mathbb{G}$, the edges of G connect the points of $\mathbb{G}$ that are at unit distance from each other. We will identify a grid graph G with the subset $\mathbb{G}$ that induces the graph.

The infinite grid $\mathbb{Z}^2$ may be viewed as the set of vertices of a tiling of the plane with unit squares. Another plane tiling with regular polygons, the tiling with equilateral triangles, defines an infinite "grid" in the same way; we call this grid infinite *triangular*. A *triangular grid graph* is a graph induced by a finite subset of the infinite triangular grid. We will use the terms triangular (square) *grid graph* and triangular (square) *grid* interchangeably. Another name for the grids used in the literature is *meshes*. Gardner [58] calls the infinite triangular grid an *isometric* grid; square grids are sometimes called *orthogonal*.

As important special cases, the classes of "thin" and "solid" (or, "simple") square grid graphs were introduced [10, 122]. A square grid is called *thin* if it contains no (simple) cycle of length 4. A square grid graph is called *solid* if all of its bounded faces are unit squares.

The HCP in square grid graphs has been the subject of extensive research [79, 105, 80, 53, 52, 122, 10]. In general, the problem is NP-complete [79, 105, 80]. It was proved that in solid square grids the HCP is polynomial [122]. It was shown [10] that in a solid square grid on N vertices there exists a tour of length at most $6N/5$ visiting all grid vertices; in *any* (not necessarily solid) square grid there exists such a tour of length $1.325N$; these tours can be computed in linear time.

A lot of effort has been devoted to establishing "simplest" classes of graphs for which the HCP remains hard. The classical result in this direction is the hardness of the problem in planar cubic graphs [60]. Another important step, also taken in [60], is establishing that the HCP in planar cubic graphs remains hard even if restricted to the graphs of girth 5. In this thesis we, in a sense, extend this result by showing that the problem remains hard in planar graphs of arbitrary girth $g \geq 6$. Since the maximum possible girth of a planar *cubic* graph is 5, instead of considering cubic graphs, we restrict our attention to planar graphs of *maximum* degree 3.

Existence of multiple Hamiltonian cycles has been the subject of extensive research too, see [64, Chapter 4] for a survey. Sufficient conditions on the degrees of the vertices of a graph are known, under which the graph, if Hamiltonian at all, contains more than one Hamiltonian cycle: any vertex has odd degree [120], any vertex has the same degree $r > 48$ [62], maximum degree is bounded from below [73], the degree of any vertex in a part (of a bipartite graph) is at least 3 [121] (and, in general, the number of Hamiltonian cycles is at least exponential in the maximum degree [121]). Thomassen [121] also considered bipartite graphs of large girth, and, as a counterpart to the above results, showed that in a Hamiltonian *cubic* graph (or when the degree of any vertex in a part is 4) the number of Hamiltonian cycles increases (at least) exponentially as a function of the girth. All these conditions bound the minimum/maximum degree of the graph vertices from below and do not restrict the graph to be planar. Here we show that there exist planar graphs of arbitrary girth $g \geq 6$, with maximum degree 3, having exactly 3 Hamiltonian cycles.

Definitions and Notation We say that a graph $G = (V, E)$ is *induced* by a set $S \subset \mathbb{R}^2$ if $V = S$, and $E = \{\{i, j\} \mid i, j \in S, |i - j| = 1\}$. Let $\mathbb{Z}_\Delta$ be the infinite triangular lattice, i.e., the set of vertices of the tiling of $\mathbb{R}^2$ with unit equilateral triangles. A *triangular grid graph*, or *triangular grid*, is a plane graph induced by a subset of vertices of $\mathbb{Z}_\Delta$. Let $G = (V, E)$ be a triangular grid without degree-1 vertices. A bounded face f of G is called a *hole* if f is not a unit equilateral triangle. Let h denote the number of holes in G. A vertex $v \in V$ is called a *cut* if its removal disconnects G; v is a *local cut* v is a cut or if the number of holes in $G \setminus v$ is less than h. A vertex $v \in V$ is called *boundary* if its degree is less than 6. The non-boundary vertices of G (degree-6 vertices) are called *internal* vertices and denoted by V_6. The subgraph B of G induced by its boundary vertices, $V \setminus V_6$, is a set of cycles $B = \{C, C_1, \dots, C_h\}$. The cycle $C \in B$ that separates G from its unbounded face is called the *outer boundary.* Each cycle in $B \setminus C = \{C_1 \dots C_h\}$ bounds a hole, i.e., is the boundary of a hole of G. Having no local cut means that the cycles in B are simple and vertex-disjoint. Let the *cost* of B be the number of edges in it.

For $g \in \mathbb{N}$ we denote by $\mathcal{G}_g$ the class of planar bipartite maximum-degree-3 graphs of girth g.

Results We show that the Hamiltonian cycle problem is NP-hard for triangular grids, even if the grid has maximum degree 4. We prove that triangular grids without local cuts are always Hamiltonian, with the exception of one special graph, "The Star of David" (Fig. 69). Our proof is constructive, which allows finding the Hamiltonian cycle in linear time. This has application in computer graphics as it gives an efficient scheme for outputting triangulation data.

We prove that for arbitrarily high g, the Hamiltonian cycle problem is NP-complete for graphs from $\mathcal{G}_g$. We also prove that for arbitrarily high g there exist graphs in $\mathcal{G}_g$ that have exactly 3 Hamiltonian cycles.

7.5.1 Hamiltonian Cycle is NP-complete for Triangular Grids

Itai et al. [79] and Papadimitriou and Vazirani [105] proved that the HCP in square grid graphs is NP-complete by a reduction from HCP in undirected planar bipartite graphs with maximum degree 3 [79]. We follow the idea of [79, 105] to show that the HCP in triangular grids is NP-complete.

Let G' be an undirected planar bipartite graph with maximum degree 3; let the nodes of G' be 2-colored "black" and "white". (We say that G' has *nodes*

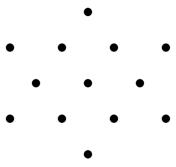


Figure 69: The only non-Hamiltonian polygonal triangular grid graph: the Star of David.

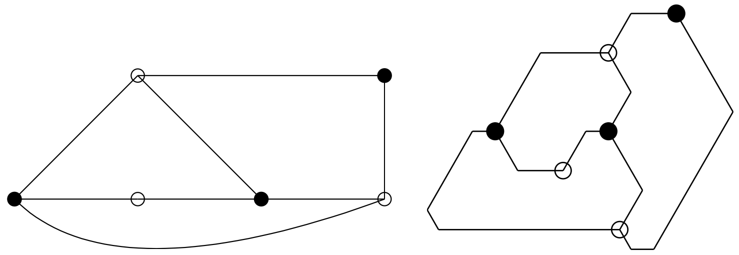


Figure 70: G' and the embedding.

and *arcs* saving the terms *vertices* and *edges* for the triangular grid graph G that we build from G' as follows.) First, G' is embedded in the plane, with the arcs drawn by paths going at 0, 60 or 120 degrees to the x-axis, so that the turn angles are 120^o at each corner along an embedded polygonal arc (Fig. 70). The embedding is then represented by a triangular grid graph G with nodes and arcs simulated by the gadgets shown in Figure 71.

In detail, the nodes are represented by the unit triangles; the arcs are simulated by "tentacles". The triangles corresponding to the black (resp., white) nodes of G' are called black (resp., white). A tentacle-arc is connected to the black triangle with a "pin" connection (Fig. 72, left) and to the white triangle with an "arm" connection (Fig. 72, right); the terms are borrowed from [105].

The only means of traversing a tentacle is either by a *return* path (Fig. 73, left) or by a (kind of a) *cross* path (Fig. 73, right). Of course, there may be many different cross paths, but the essential difference between the return and the cross paths is that the former connects the tentacle vertices aligned along a line, while the latter "jumps back and forth" between the two lines that bound the tentacle. The idea of the difference is that a cross path connects the two node gadgets at its ends, while a return path just traverses the vertices in the

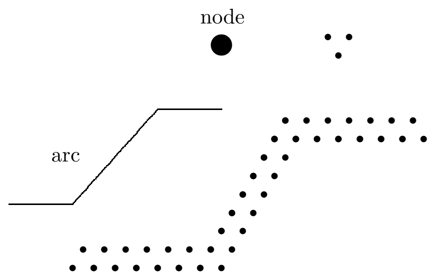


Figure 71: The gadgets.

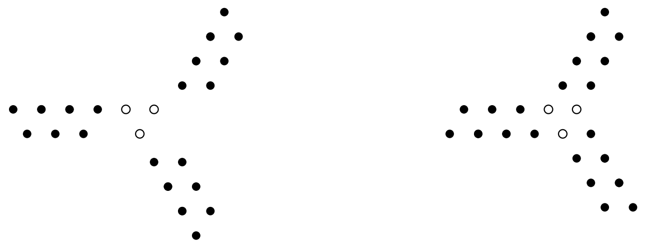


Figure 72: A "pin" connection (left) and an "arm" connection (right). The node gadgets are shown with hollow circles.

tentacle, returning to the same end from which it started.

Theorem 7.30. *The HCP for triangular grid graphs is NP-complete.*

Proof. If G' has a Hamiltonian cycle, then G has one, which traverses the black and white triangles of G in the order of the corresponding nodes of G' in the cycle. It traverses by cross paths the tentacles that correspond to arcs in the cycle. The remaining tentacles are picked up by return paths from the adjacent white triangles.

Conversely, any Hamiltonian cycle $\mathcal{C}$ of G comes from a Hamiltonian cycle of G' in this way. Indeed, it is not hard to see, by inspection of Fig. 72, that in $\mathcal{C}$ any triangle, representing a node of G', is attached to exactly two cross paths. □

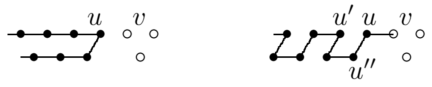


Figure 73: The paths.

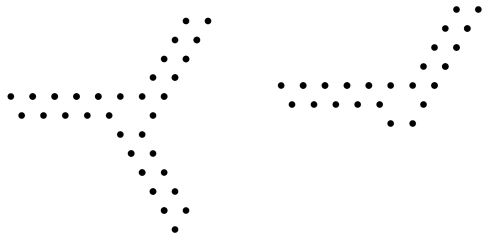


Figure 74: Left: Modified white triangle. Right: Modified turn of a tentacle.

Papadimitriou and Vazirani [105] also proved that the HCP in square grid graphs is NP-complete even when restricted to graphs of maximum degree 3; Buro [25] gave an alternative proof. Here we prove that the HCP in triangular grids is NP-complete even when restricted to grids of maximum degree 4.

The graph G constructed in the proof of Theorem 7.30 has certain vertices of degree 5, namely, the vertices of the white triangles and the inner points of the angles of the tentacles. Figure 74 shows how the construction may be modified so that the resulting graph has vertices of degree 4 or less.

Theorem 7.31. *The HCP for triangular grid graphs with maximum degree 4 is NP-complete.*

7.5.2 Triangular Grids without Local Cuts are Hamiltonian

Our proof of the hardness of the HCP in triangular grids (given in the previous subsection) relied on the grid having local cuts: such are, e.g., the "black" ends of the tentacles — the "pin" connections (Fig. 71, right). It turns out that having local cuts is crucial for the hardness of the problem: as we prove below, the HCP is polynomial in the triangular grids *without* local cuts. In fact, the connectivity of a triangular grid is so high that, with the exception of one particular graph (which we call the "Star of David", Fig. 69), all triangular grids without local cuts are Hamiltonian. Our proof is constructive and can be turned into a linear-time algorithm, producing a Hamiltonian cycle though the grid.

The crucial observations that we use are as follows: 1) One can attach to the cycles in B all internal vertices at the cost of 1 per vertex; this way a cycle cover of G is obtained, in which the cycles are vertex-disjoint. 2) A cover B of G by vertex-disjoint cycles may be modified so that for any cycle $C_i \in B$ there exists a cycle $C_j \in B$, "facing" C_i (Fig. 75) — thus all the cycles may be spliced into one, Hamiltonian, cycle through G. We formalize and prove

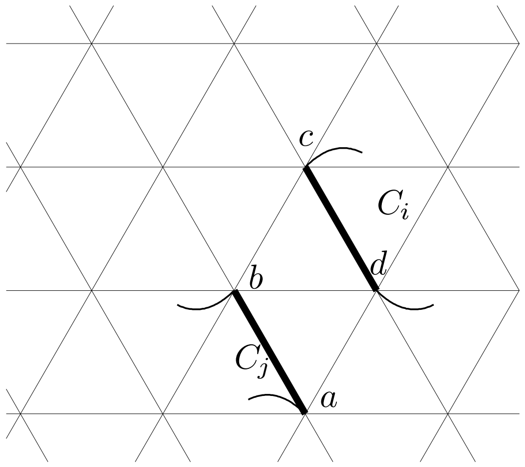


Figure 75: C_i and C_j face each other: they can be spliced together by flipping the opposite edges of the rhombus *abcd*.

these observations in the next two lemmas.

Lemma 7.32. *Unless G is the Star of David, the cycles in B can be modified into a set of cycles that visit all vertices in V_6. The cost of the modification is 1 per vertex of V_6.*

Proof. We modify B by consistently applying three types of local modifications, which we call the L-, V- and Z-modifications. Let $B' = \{C', C'_1, \ldots, C'_h\}$ be the cycles at any particular stage of the modification; we maintain the invariant that C' is a simple cycle within G such that all of the vertices of G that have not been visited by the cycles in B' (i.e., $V \setminus C' \setminus C'_1 \ldots \setminus C'_h$) are inside C'. Each modification adds one new vertex to a cycle in B'. The V-modification is applied only when L cannot be applied; the Z-modification is applied only when no other modification can be applied. The modifications are "monotone" in that each modification will result in B' visiting a superset of the vertices that it previously visited.

We now describe the modifications. Let $v \in V_6 \setminus B'$ be an unvisited vertex. The L-modification is applied as long as there exists a unit equilateral triangle abv such that ab is an edge of a cycle in B' (Fig. 76). The V-modification is applied only when L cannot be applied and B' goes around v like in Figure 77, left; the modified B' is shown in Fig 77, right. Finally, the Z-modification is applied only when none of L, V can be applied and B' goes around v like in Figure 78, left; the modified B' is shown in Fig 78, right.

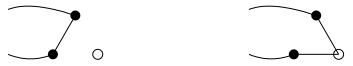


Figure 76: The L-modification. v is the hollow circle.

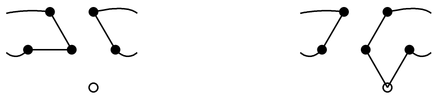


Figure 77: The V-modification. v is the hollow circle.

We introduce some simple definitions. Let $u \in B'$ be a vertex visited by a cycle in B'. We say that u is a *wide* (resp., *sharp*) *wedge* if B' makes a 120^o (resp., 60^o) turn at u (Fig. 79).

We now proceed to showing that all vertices in V_6 can be attached to B' as claimed. Suppose that at some stage none of the modifications L, V, or Z can be applied, but B' does not yet go through all vertices in V_6. Then, since G is connected, there exists a vertex $v \in V_6 \setminus B'$ such that at least one of the neighbors of v (say, u) is a vertex of B'. Observe that the degree of v in G is 6, for otherwise v is a boundary vertex and a cycle in B has been going through v from the very beginning.

Since L cannot be applied, none of the edges of the hexagon that "surrounds" v is in B' (Fig. 80, left). Since u is in B', at least one of the edges 1,2 in Fig. 80, left, must be in B'. Since L cannot be applied, at least one of the vertices that adjacent both to v and u must be in B' (say, the edge 1 in Fig. 80, left, is in B', so the vertex s is in B' (Fig. 80, right)).

Consider three cases:

Case I: s is a sharp wedge like in Fig. 81, left. Then V can be applied to attach v to B' (Fig. 81, right).

Case II: s is a sharp wedge like in Fig. 82, left. Then Z can be applied

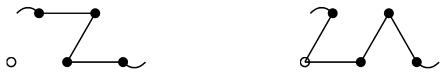


Figure 78: The Z-modification. v is the hollow circle.

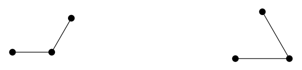


Figure 79: A wide (left) and a sharp (right) wedges.

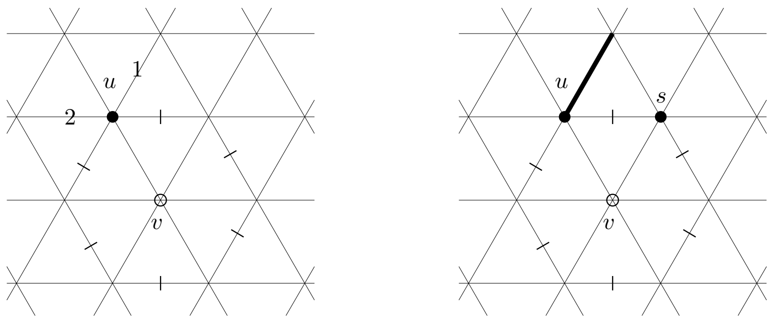


Figure 80: Left: none of the crossed edges may be in B'. At least one of the edges 1,2 (say, 1) is in B'. Right: then $s \in B'$.

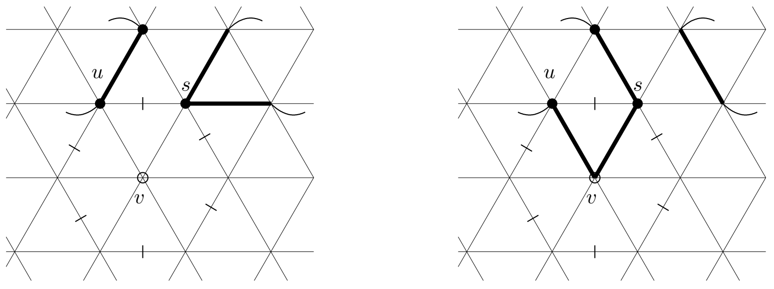


Figure 81: s is a sharp wedge, and a V may be applied.

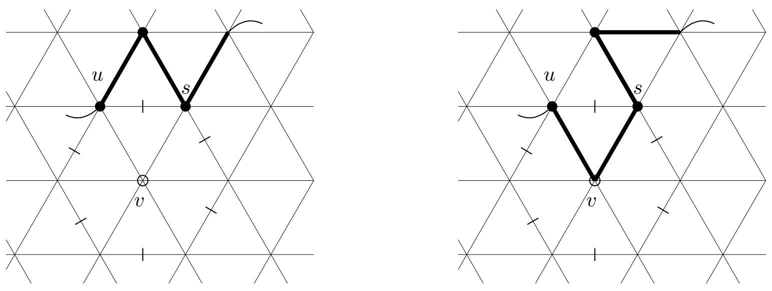


Figure 82: s is a sharp wedge, and a Z may be applied.

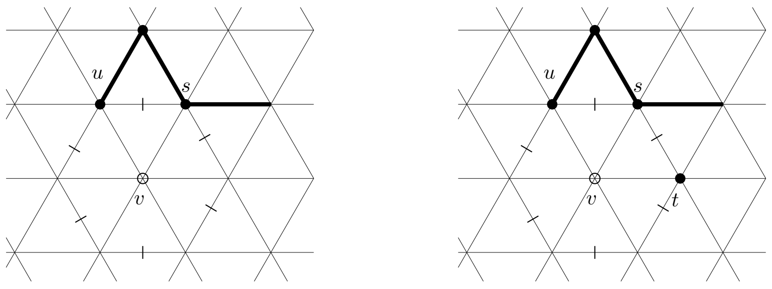


Figure 83: Left: s is a wide wedge. Right: then $t \in B'$, for otherwise L could be applied.

to attach v to B' (Fig. 82, right).

Case III: s is a wide wedge (Fig 83, left). Then, since L can not be applied, the vertex t (Fig. 83, right), adjacent to both v and s, is in B'.

Now, by considering the same three cases of how B' goes through t, one may conclude that, unless t is a wide wedge (Case III), v can be attached to B' at the cost of 1. But if t is a wide wedge (Fig. 84, left), then, since L can not be applied, the vertex x, adjacent to both v and t, is in B' (Fig. 84, right). Considering the three cases of how B' goes through x, we conclude that x is a wide wedge too, the vertex, adjacent to both v and x is in B', and is also a wide wedge. Continuing, we see that if v can not be attached to B' at the cost of 1, the part of B' that goes around v is one cycle, C, which is the boundary of the Star of David. Since v is a vertex of G, the cycle C does not surround a hole in G, thus it is the outer boundary of G, and G is the Star of David. □

An interesting corollary from the above lemma is that G admits a cycle

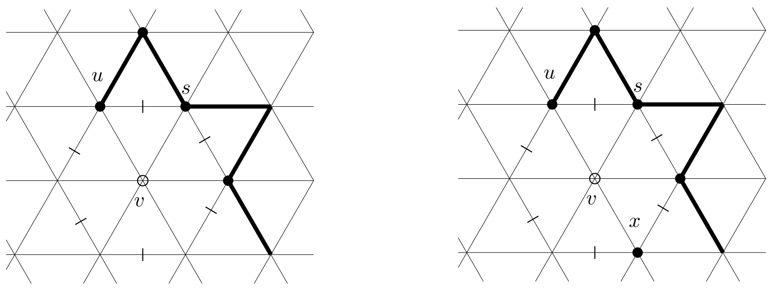


Figure 84: Left: t is a wide wedge. Right: then $x \in B'$, for otherwise L could be applied.

cover by *vertex-disjoint* cycles.

Corollary 7.33. *Unless G is the Star of David, G admits a cycle cover of cost $|V|$.*

We prove now that any cycle cover of G by vertex-disjoint cycles can be spliced together into one, Hamiltonian, cycle through G. We do it by showing that the cycle cover may be modified, by local modifications, into another cycle cover, in which there exist two cycles that "face" each other.

Definition 7.34. *Let C_i, C_j be two cycles in G. We say that C_i, C_j* face *each other if there exists a unit rhombus $abcd$, $a, b, c, d \in V$ with $ab \in C_i$, $cd \in C_j$ (see Fig. 75).*

Lemma 7.35. *Let B be a set of vertex-disjoint cycles going through all vertices of G. Then there exists two cycles in B that can be modified into cycles that face each other.*

Proof. Since G is connected, there must exist two vertices, u and v, adjacent in G, belonging to different cycles, say $u \in C_i$, $v \in C_j$. Since G is local-cut-free, one of the nodes of the grid, adjacent to both u and v, must be in G (Fig. 85). In other words, there must exist a unit equilateral triangle uvx within G whose vertices are visited by more than one cycle in B.

Consider two cases:

Case I: *one of the edges of the triangle belongs to a cycle in B, Fig. 86, left.* Suppose that $ux \in C_i$, $v \in C_j$. If any of the crossed edges in Fig. 86, left, is in B, then the cycles C_i and C_j already face each other without any modifications, so, suppose, the crossed edges are not in B. This leaves

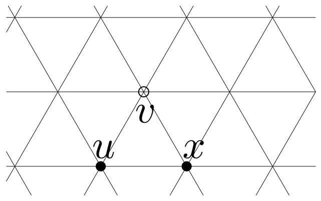


Figure 85: $u \in C_i$, $v \in C_j$. $x \in G$.

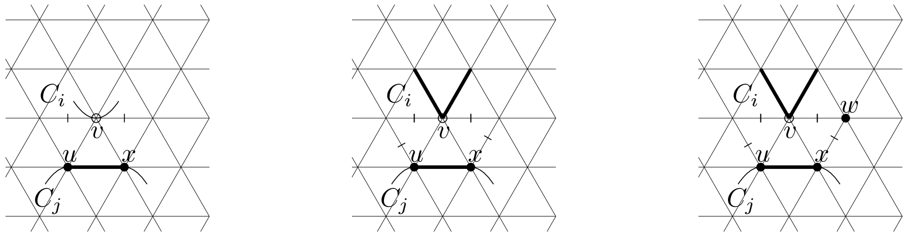


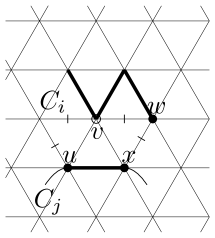
Figure 86: Left: $ux \in C_i$, $v \in C_j$, crossed edges are not in B. Center: the edges of C_j adjacent to v may be deduced; crossed edges are not in B. Right: $w \in G$.

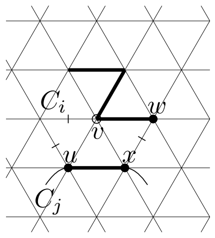
only two edges of G that could be adjacent to v in B (Fig. 86, center). This, in turn, implies that if any of the edges, crossed in Fig. 86, center, are in B, then C_i and C_j face each other; so suppose the crossed edges are not in B.

For v not to be a local cut, at least one of the vertices of the grid that are at distance 1 from v and are at the same horizontal line as v, must be in G; suppose, without loss of generality, that it is a vertex w to the right of v (Fig. 86, right). If B goes through w as in Fig. 87, left, a Z-modification may be applied to C_j to have C_i and modified C_j face each other (Fig. 87, center). So, we may assume that the crossed edges in Fig. 87, right, are not in B. Let's consider how B may go through w.

Suppose that B goes through w as in Fig. 88, left or Fig. 88, center. If $w \in C_k \neq C_i$, then C_i and C_k already face each other; so, suppose that $w \in C_i$. Then the modifications as in Fig 89, left and Fig 89, center, lead to modified C_i and C_j facing each other. Finally, if B goes through w as in Fig. 88, right, a modification as in Fig. 89, right, leads to the desired result.

Case II: *none of the edges of the triangle belongs to a cycle in B.* In other





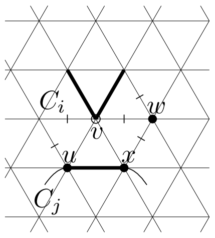
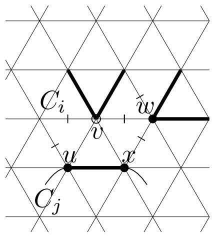
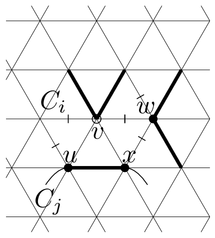


Figure 87: Left and center: A Z leads to the cycles facing each other. Right: so, assume all crossed edges are not in B.





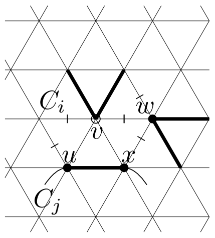
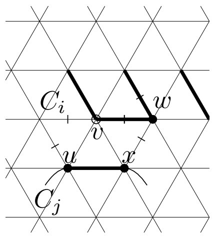
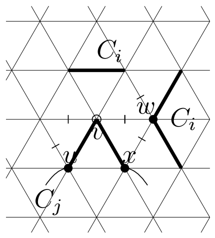


Figure 88: Different ways in which the B may go through w. Left and center: If $w \notin C_i$, we already have facing edges.





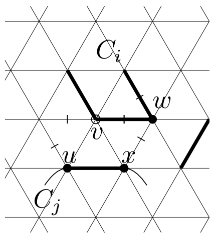


Figure 89: The modifications depending on how B goes through w in Fig. 88.

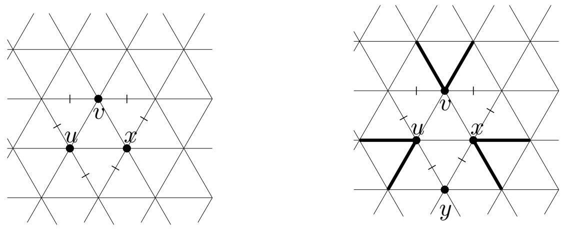


Figure 90: $C_u \neq C_v \neq C_x$.

words, u, v, x belong to different cycles, say C_u, C_v, C_x. If one of the edges, crossed in Fig. 90, left, is in B, then we are in Case I; so suppose that none of the crossed edges is in B. This leaves, for each of u, v, x, only two edges of the grid that are possibly adjacent to the vertex in B (Fig. 90, right). Since G has no local cuts, at least 1 other vertex adjacent to a crossed edge is in G (in fact, at least 2). Suppose $y \in G$ (Fig. 90. right). Now, no matter how B visits y, we can find facing cycles. Indeed, if $y \in C_u$ or $y \in C_x$, then we are in Case I. Otherwise, the cycle, that y belongs to, faces both C_u and C_x.

□

Thus, starting from the boundary cycles of G, one may apply the modifications as in Lemma 7.32 and then as in Lemma 7.35 to get a Hamiltonian cycle through G. This is our main result:

Theorem 7.36. *Except for the Star of David (Fig. 69), any triangular grid without local cuts is Hamiltonian and a Hamiltonian cycle in it may be found in linear time.*

Some work in computer graphics focused on finding long cycles through triangular meshes (see, e.g., [51, 26, 42]). We feel that the following corollary may have important applications.

Corollary 7.37. *Let M be a triangulated manifold, whose connectivity is at least that of the triangular grid. Then, if M has no local cuts, there exists a Hamiltonian cycle through the vertices of M, and the cycle can be found in linear time.*

7.5.3 Hamiltonian Cycles in High-Girth Graphs

Our results in this subsection come from an attempt to prove the hardness of the HCP in "hexagonal" grid graphs. Similarly to the square and triangular

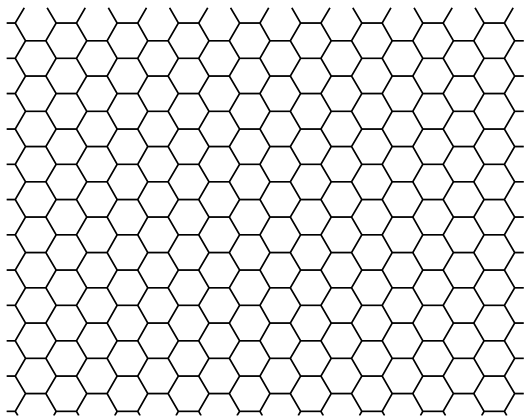


Figure 91: The infinite hexagonal grid.

grids, we may consider tiling of the plane with regular unit-side hexagons (Fig. 91); we say that the vertices of the tiling form the *infinite* hexagonal grid. A *hexagonal grid graph*, or *hexagonal grid* is induced by a subset of the vertices of the infinite hexagonal grid: the vertices of the graph are the grid vertices in the subset, the edges connect vertices that are at distance 1.

To show that the HCP in high-girth graphs is NP-complete we, as in Section 7.5.1, follow the idea of [79, 105] and reduce from the HCP in planar bipartite graphs of maximum degree 3 [79]. Let G' be an undirected planar bipartite graph with maximum degree 3; let the nodes of G' be 2-colored "black" and "white". As before, G' is embedded in the plane, with the arcs drawn by paths going at 0, 60 or 120 degrees to the x-axis, so that the turn angles are 120^o at each corner along an embedded polygonal arc (Fig. 92). Then, we try to represent the embedding by a hexagonal grid graph. Although we do not succeed in it, we use the gadgets of the (incomplete) representation to construct the high-girth graph G such that G is Hamiltonian if and only if G' is.

As in [79, 105] and Section 7.5.1, the arcs of G' are represented by *tentacles* (Fig. 93). The only means of traversing a tentacle is either by a *cross* path (Fig. 94) or by a *return* path (Fig. 95). A cross path connects the two node gadgets at its ends, while a return path just traverses the vertices in the tentacle, returning to the same end from which it started.

We represent a white node of G' by the *white-node gadget* (Fig. 96). As soon as two edges that pick up the central vertex of the gadget are decided, the gadget becomes incident to two cross paths and one return path (Fig. 97).

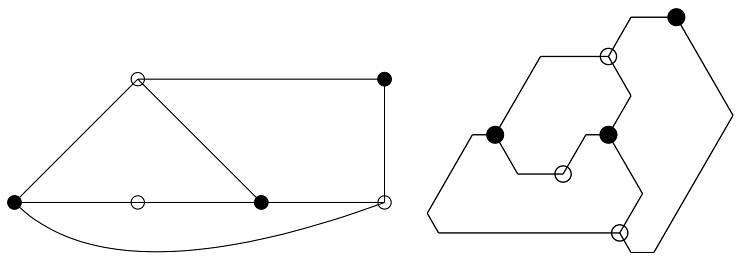


Figure 92: G' and the embedding.

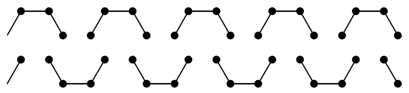


Figure 93: The tentacle.

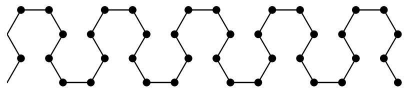


Figure 94: The cross path.

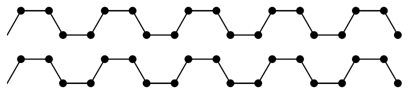


Figure 95: The return path.

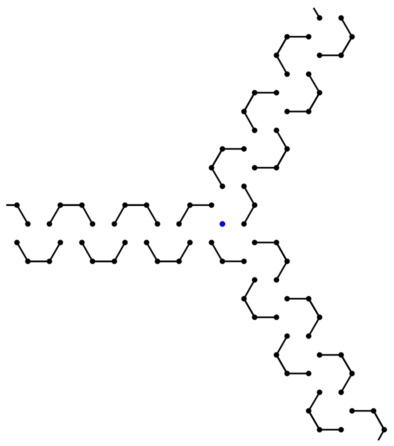


Figure 96: The white-node gadget.

A black node of G' is represented by the *black-node gadget* (Fig. 98). As soon as two edges that pick up the central vertex of the gadget are decided, the gadget becomes incident to two cross paths and have one return path "close" inside the gadget (Fig. 99).

Observe (Figs. 97, 99) that the return paths are consistent in that they "originate" at a white node and "close" at a black node.

To complete the reduction it is necessary to show how to *turn* the tentacles. Unfortunately, turning our tentacles within the hexagonal grid is impossible (due to a parity conflict). However, if we stop working in a hexagonal grid and consider graph G just as an abstract planar graph, then turning is not an issue at all (Fig. 100).

Lemma 7.38. *The HCP in planar graphs of maximum degree 3 and girth 6 is NP-complete.*

Proof. The "building blocks" of the graph G — tentacles and node gadgets — come from the hexagonal grid, which has maximum degree 3 and girth 6. (The only part of G that did not come from the hexagonal grid is the turn of a tentacle (Fig. 100), but, clearly, it did not introduce vertices of higher degree or short cycles.) Thus, the maximum degree of G is 3 and its girth is 6.

By construction, Hamiltonian cycles in G are in one-to-one correspondence with Hamiltonian cycles in G'. Indeed, as in [79, 105] and Section 7.5.1, cross paths in G correspond to the edges of G' that are in a cycle; the edges of

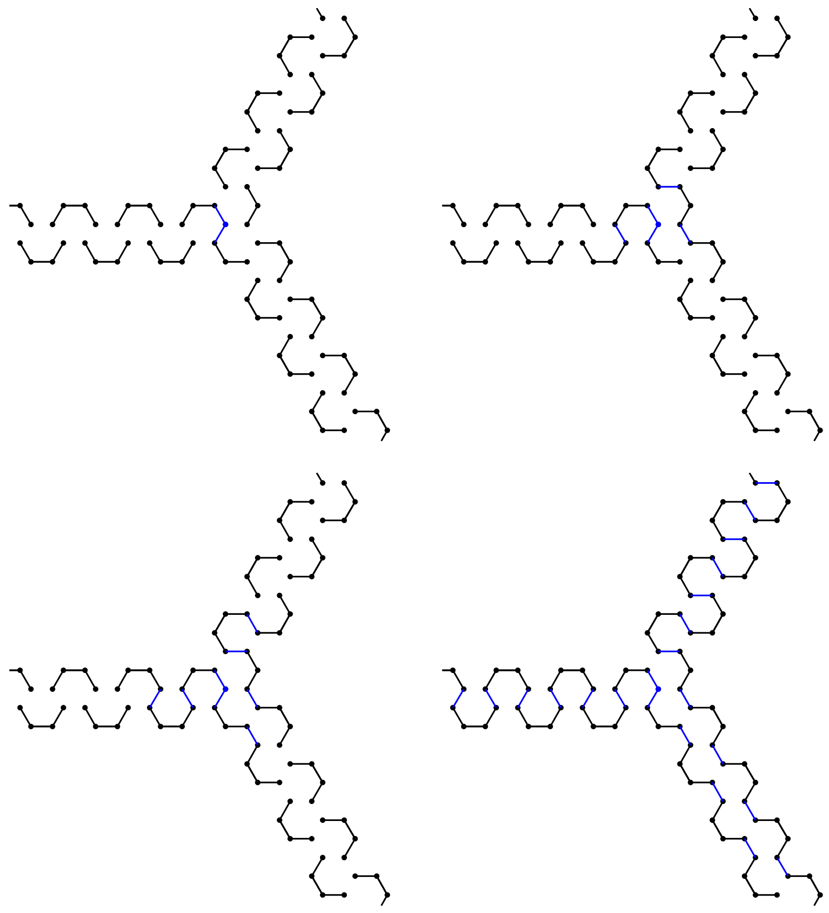


Figure 97: A white-node gadget is incident to two cross paths and one return path.

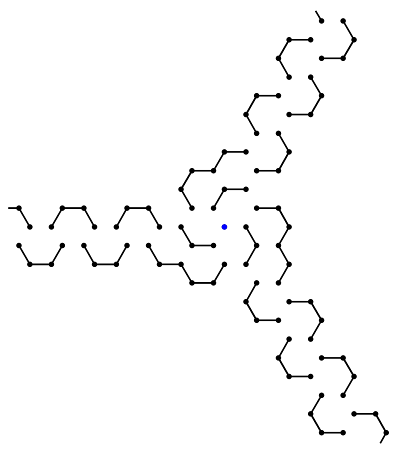


Figure 98: The black-node gadget.

G that are not, are picked up by return paths from the adjacent white-node gadgets. □

Theorem 7.39. *For arbitrary $g \geq 6$, the HCP in planar graphs of maximum degree 3 and girth g is NP-complete.*

Proof. Let E_2 be the edges of G that are adjacent to degree-2 vertices; call these edges *black*. (In fact, these are the black edges that are drawn in Figs. 93, 96, and 98; black edges must participate in any Hamiltonian cycle in G.) Replace every edge in E_2 with a path of length g; each vertex in the path is of degree 2. Make every tentacle long, so that it has at least g "wiggles" (see Fig. 93). Call the resulting graph G_g. By construction, G_g is planar and its maximum degree is 3.

Claim 7.40. *The girth of G_g is at least g.*

Proof. Let E_1 be the edges of G that are not black (so, call them *white*). The edges of G_g may also be called white and black: white edges are those that correspond to white edges of G, and black — those that appeared as a result of subdividing black edges of G. Any cycle in G_g that uses a black edge has length more than g. It is not hard to see (Figs. 93, 96, and 98) that any cycle within a tentacle or a node gadget has to use black edges. Finally, a cycle in G_g, consisting solely of white edges and going through two or more node gadgets, has to traverse a tentacle and thus, has length greater than g. □

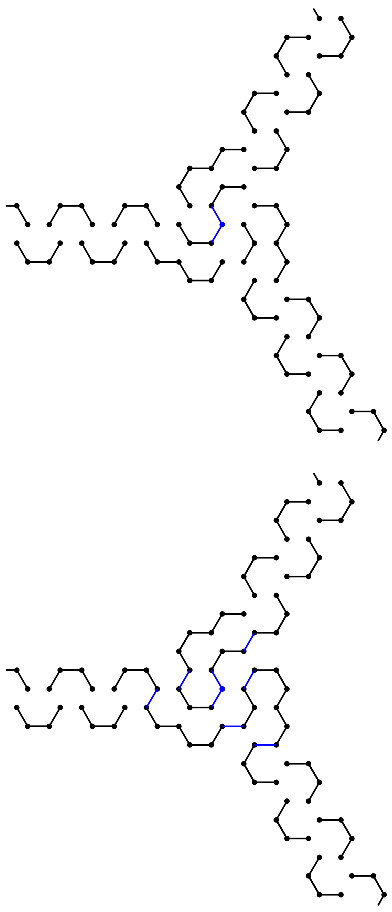


Figure 99: A black-node gadget is incident to two cross paths and one return path.

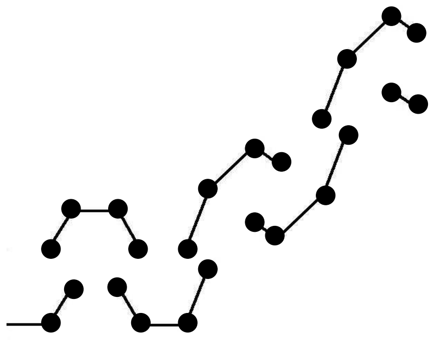


Figure 100: Turning a tentacle is not an issue in the abstract graph setting.

Hamiltonian cycles in G_g map one-to-one into Hamiltonian cycles in G, and, thus, into Hamiltonian cycles in G'. Hence, G_g is Hamiltonian if and only if G' is. $\square$

As a by-product of our reduction we obtain, for any $g \geq 6$, a one-to-one mapping between Hamiltonian cycles in planar bipartite graphs of maximum degree 3 and Hamiltonian cycles in planar girth-g graphs of maximum degree 3. This allows one to reason about Hamiltonian cycles in the latter in terms of the Hamiltonian cycles in the former. For instance,

Theorem 7.41. *For any $g \geq 6$ there exist planar girth-g graphs of maximum degree 3 that contain exactly 3 Hamiltonian cycles.*

Proof. The (planar, bipartite, maximum-degree-3) graph G' in Fig. 101 has exactly three Hamiltonian cycles (Fig 102). Thus, the high-girth graph, constructed from G' by the procedure in our reduction, also has exactly three Hamiltonian cycles. $\square$

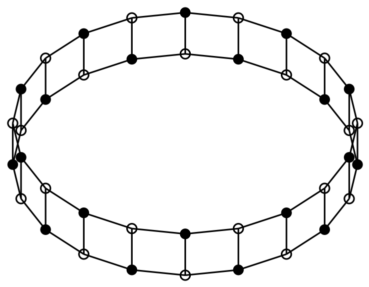


Figure 101: G'.

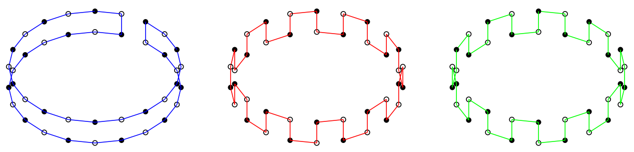


Figure 102: The 3 Hamiltonian cycles in G'.

8 Conclusion and Open Problems

We considered the problem of finding shortest non-crossing thick paths in polygonal domains. We presented polynomial-time algorithms for the problem in simple polygons, and showed NP-hardness for polygonal domains with holes. For the latter case we suggested an algorithm, whose running time is exponential in the number of holes, but is polynomial in the other input parameters.

We studied minimum-cost flows in geometric environments. We formulated and proved a continuous analog of the Flow Decomposition Theorem: a minimum-cost flow may be decomposed into a set of (thick) paths. The theorem allowed us to apply our algorithms on finding thick paths to obtain the first algorithmic results for the minimum-cost flow problem.

For the problem of routing the maximum number of thick paths, we proved a continuous Megner-type result, the Discrete Continuous MaxFlow-MinCut Theorem: the maximum number of thick paths in a polygonal domain equals to the shortest path in the thresholded critical graph of the domain. We also gave algorithms for finding monotone paths and flows.

The ATM motivates studying dynamic and stochastic versions of path planning. It remains open to solve the flow/paths problems in higher dimensions, in particular, in the (x, y, t)-space. Our algorithms work in the rectilinear world, but do rectilinear paths/flows approximate arbitrary ones well? The first steps in this direction would be to consider the problems on two-dimensional manifolds in 3D, and on unions of such manifolds.

Another relevant issue is estimating the capacity of an airspace close to an airport. This brings up the problem of finding a maximum flow in a *circular annulus*. By bridging each hole, in turn, to the inner circle of the annulus, we can compute the maxflow in $O(n^3)$ time. Is $O(n^2)$ algorithm possible?

It seems, that despite its importance, the problem of finding a thick wire (a non-self-overlapping thick path) has not been solved.

Finally, an important open problem is that of finding optimal *constrained* paths, e.g., polygonal paths with no sharp angles.

References

[1] http://www.centralvacuumstores.com/vacpan.htm.

[2] http://www.plowsunlimited.com/snow_melters.htm.

[3] P. K. Agarwal, A. Efrat, and M. Sharir. Vertical decomposition of shallow levels in 3-dimensional arrangements and its applications. *SIAM J. Comput.*, 29:912–953, 1999.

[4] P. K. Agarwal, P. Raghavan, and H. Tamaki. Motion planning for a steering-constrained robot through moderate obstacles. In *Proc. 27th Annu. ACM Sympos. Theory Comput.*, pages 343–352, 1995.

[5] P. K. Agarwal and M. Sharir. Arrangements and their applications. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 49–119. Elsevier Science B.V. North-Holland, Amsterdam, 2000.

[6] A. Aggarwal, M. M. Klawe, S. Moran, P. W. Shor, and R. Wilber. Geometric applications of a matrix searching algorithm. In *Proc. 2nd Annu. ACM Sympos. Comput. Geom.*, pages 285–292, 1986.

[7] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall, Englewood Cliffs, NJ, 1993.

[8] E. J. Anderson, P. Nash, and A. B. Philpott. A class of continuous network flow problems. *Math Oper Res*, 7(4):501–514, 1982.

[9] E. M. Arkin, R. Connelly, and J. S. B. Mitchell. On monotone paths among obstacles, with applications to planning assemblies. In *Proc. 5th Annu. ACM Sympos. Comput. Geom.*, pages 334–343, 1989.

[10] E. M. Arkin, S. P. Fekete, and J. S. B. Mitchell. Approximation algorithms for lawn mowing and milling. *Computational Geometry*, 17(1-2):25–50, 2000.

[11] E. M. Arkin, M. Held, and C. L. Smith. Optimization problems related to zigzag pocket machining. *Algorithmica*, 26(2):197–236, 2000.

[12] E. M. Arkin, J. S. B. Mitchell, and C. D. Piatko. Bicriteria shortest path problems in the plane. In *Proc. 3rd Canad. Conf. Comput. Geom.*, pages 153–156, 1991.

[13] D. S. Atkinson and P. M. Vaidya. Using geometry to solve the transportation problem in the plane. *Algorithmica*, 13(5):442–461, 1995.

[14] F. Aurenhammer. Voronoi diagrams: A survey of a fundamental geometric data structure. *ACM Comput. Surv.*, 23(3):345–405, Sept. 1991.

[15] O. Bastert and S. P. Fekete. Geometric wire routing. Technical Report 332, Zentrum für Angewandte Informatik, 1998.

[16] M. S. Bazaraa, J. J. Jarvis, and H. F. Sherali. *Linear Programming and Network Flows*. John Wiley & Sons, New York, NY, USA, second edition, 1990.

[17] S. Bereg and D. G. Kirkpatrick. Curvature-bounded traversals of narrow corridors. In *Symposium on Computational Geometry*, pages 278–287, 2005.

[18] S. Bespamyatnikh. Computing homotopic shortest paths in the plane. *J. Algorithms*, 49(2):284–303, 2003.

[19] S. Bespamyatnikh. Encoding homotopy of paths in the plane. In *Proc. of the 6th Latin American Theoretical INformatics (LATIN'04)*, LNCS 2976, pages 329–338, 2004.

[20] J.-D. Boissonnat, J. Czyzowicz, O. Devillers, J.-M. Robert, and M. Yvinec. Convex tours of bounded curvature. *Comput. Geom. Theory Appl.*, 13:149–159, 1999.

[21] J.-D. Boissonnat and S. Lazard. A polynomial-time algorithm for computing a shortest path of bounded curvature amidst moderate obstacles. In *Proc. 12th Annu. ACM Sympos. Comput. Geom.*, pages 242–251, 1996.

[22] S. Boyd and L. Vandenberghe. *Convex optimization.* Cambridge University Press, 2004.

[23] D. Bremner, J. O'Rourke, and T. Shermer. Motion planning amidst movable square blocks is PSPACE complete. Draft, June, 1994.

[24] D. Burago, D. Grigoriev, and A. Slissenko. Approximating shortest path for the skew lines problem in time doubly logarithmic in 1/epsilon. *Theor. Comput. Sci.*, 315(2-3):371–404, 2004.

[25] M. Buro. Simple amazons endgames and their connection to Hamilton circuits in cubic subgrid graphs. *Lecture Notes in Computer Science*, 2063:250–261, 2001.

[26] A. Bushan, P. Diaz-Gutierrez, D. Eppstein, and M. Gopi. Single triangle strip and loop on manifolds with boundaries. In *Proc. 19th Brazilian Symp. Computer Graphics and Image Processing*, pages 221–228.

[27] S. Cabello, Y. Liu, A. Mantler, and J. Snoeyink. Testing homotopy for paths in the plane. In *SCG '02: Proceedings of the eighteenth annual symposium on Computational geometry*, pages 160–169, New York, NY, USA, 2002. ACM Press.

[28] D. Z. Chen, O. Daescu, and K. S. Klenk. On geometric path query problems. *Internat. J. Comput. Geom. Appl.*, 11(6):617–645, 2001.

[29] L. P. Chew. Planning the shortest path for a disc in $O(n^2 \log n)$ time. In *Proc. SoCG'85*, pages 214–220, 1985.

[30] F. Chin, J. Snoeyink, and C. A. Wang. Finding the medial axis of a simple polygon in linear time. *Discrete Comput. Geom.*, 21(3):405–420, 1999.

[31] S. Cohen and L. Guibas. The earth mover's distance: Lower bounds and invariance under translation. Technical Report CS-TR-97-1597, 1997.

[32] R. Cole and A. Siegel. River routing every which way, but loose. In *Proc. 25th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 65–73, 1984.

[33] J. Culberson. Sokoban is PSPACE-complete. In E. Lodi, L. Pagl, and N. Santoro, editors, *Proc. Internat. Conf. Fun with Algorithms*, pages 65–76, Elba, Italy, June 1998. Carelton Scientific.

[34] J. Culberson. Private communication, 2003.

[35] T. Dayan. *Rubber-Band Based Topological Router*. PhD thesis, UC Santa Cruz, 1997.

[36] M. de Berg, M. van Kreveld, B. J. Nilsson, and M. H. Overmars. Shortest path queries in rectilinear worlds. *Internat. J. Comput. Geom. Appl.*, 2(3):287–309, 1992.

[37] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications.* Springer-Verlag, Berlin, 1997.

[38] E. Demaine, M. Demaine, and J. O'Rourke. PushPush and Push-1 are NP-hard in 2D. In *Proc. 12th Canad. Conf. Comput. Geom.*, pages 211–219, Fredericton, New Brunswick, Canada, August 16–18 2000.

[39] E. Demaine, R. Hearn, and M. Hoffmann. Push-2-F is PSPACE-complete. In *Proc. 14th Canad. Conf. Comput. Geom.*, pages 31–35, Lethbridge, Alberta, Canada, August 12–14 2002.

[40] E. Demaine and M. Hoffmann. Pushing blocks is NP-complete for non-crossing solution paths. In *Proc. 13th Canad. Conf. Comput. Geom.*, pages 65–68, Waterloo, Ontario, Canada, August 13–15 2001.

[41] E. D. Demaine, M. L. Demaine, M. Hoffmann, and J. O'Rourke. Pushing blocks is hard. *Comput. Geom. Theory Appl.*, 26(1):21–36, 2003.

[42] E. D. Demaine, D. Eppstein, J. Erickson, G. W. Hart, and J. O'Rourke. Vertex-unfoldings of simplicial manifolds. In *SCG '02: Proceedings of the eighteenth annual symposium on Computational geometry*, pages 237–243, New York, NY, USA, 2002. ACM Press.

[43] E. Demanie. http://theory.lcs.mit.edu/∼edemaine/pushingblocks/.

[44] D. P. Dobkin and D. L. Souvaine. Computational geometry in a curved world. *Algorithmica*, 5:421–457, 1990.

[45] D. Dor and U. Zwick. Sokoban and other motion planning problems. *Comput. Geom. Theory Appl.*, 13(4):215–228, 1999.

[46] Z. Drezenr. *Facility location: a survey of applications and methods.* Springer, New York, 1995.

[47] M. Dror, A. Efrat, A. Lubiw, and J. S. B. Mitchell. Touring a sequence of polygons. In *STOC '03: Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 473–482, New York, NY, USA, 2003. ACM Press.

[48] C. A. Duncan, A. Efrat, S. G. Kobourov, and C. Wenk. Drawing with fat edges. In *GD '01: Revised Papers from the 9th International Symposium on Graph Drawing*, pages 162–177, London, UK, 2002. Springer-Verlag.

[49] A. Efrat, S. Kobourov, M. Stepp, and C. Wenk. Growing fat graphs. In *SCG '02: Proceedings of the eighteenth annual symposium on Computational geometry*, pages 277–278, New York, NY, USA, 2002. ACM Press.

[50] A. Efrat, S. G. Kobourov, and A. Lubiw. Computing homotopic shortest paths efficiently. In *Proceedings of the 10th Annual European Symposium on Algorithms*, pages 411–423, London, UK, 2002. Springer-Verlag.

[51] D. Eppstein and M. Gopi. Single-strip triangulation of manifolds with arbitrary topology. In *SCG '04: Proceedings of the twentieth annual symposium on Computational geometry*, pages 455–456, New York, NY, USA, 2004. ACM Press.

[52] H. Everett. Finding hamiltonian paths in nonrectangular grid graphs. *Congressus Numerantium*, 53:185–192, 1986.

[53] S. P. Fekete. *Geometry and the Travelling Salesman Problem.* Ph.D. thesis, Department of Combinatorics and Optimization, University of Waterloo, Waterloo, ON, 1992.

[54] S. Foldes. Conditions for the separability of objects in two-dimensional velocity fields. *Canad. Math. Bull.*, 35(4):484–491, 1992.

[55] A. Fournier and D. Y. Montuno. Triangulating simple polygons and equivalent problems. *ACM Trans. Graph.*, 3(2):153–174, 1984.

[56] K. Fujimura. Time-minimal paths amidst moving obstacles in three dimensions. *Theor. Comput. Sci.*, 270(1-2):421–440, 2002.

[57] S. Gao, M. Jerrum, M. Kaufman, K. Mehlhorn, and W. Rülling. On continuous homotopic one layer routing. In *SCG '88: Proceedings of the fourth annual symposium on Computational geometry*, pages 392–402, New York, NY, USA, 1988. ACM Press.

[58] M. Gardner. *Knotted Doughnuts and Other Mathematical Entertainments.* W. H. Freeman, New York, 1986.

[59] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, New York, NY, 1979.

[60] M. R. Garey, D. S. Johnson, and R. E. Tarjan. The planar Hamiltonian circuit problem is NP-complete. *SIAM J. Comput.*, 5(4):704–714, 1976.

[61] L. Gewali, A. Meng, J. S. B. Mitchell, and S. Ntafos. Path planning in $0/1/\infty$ weighted regions with applications. *ORSA J. Comput.*, 2(3):253–272, 1990.

[62] M. Ghandehari and H. Hatami. A note on independent dominating sets and second hamiltonian cycles. (submitted).

[63] J. Glimm and D. H. Sharp. Complex fluid mixing flows: Simulation vs. theory vs. experiment. *SIAM News*, 39(5), June 2006.

[64] R. Gould. Advances on the Hamiltonian problem — a survey. *Graphs Combin*, 19:7–52, 2003.

[65] L. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. Tarjan. Linear time algorithms for visibility and shortest path problems inside simple polygons. In *SCG '86*, pages 1–13, New York, NY, USA, 1986. ACM Press.

[66] R. Hearn and E. Demaine. The nondeterministic constraint logic model of computation: Reductions and applications. In *Proc. 29th International Colloquium on Automata, Languages and Programming*, volume 2380 of *Lecture Notes in Computer Science*, pages 401–413, Malaga, Spain, July 8–13 2002.

[67] M. Held. *On the Computational Geometry of Pocket Machining*, volume 500 of *Lecture Notes Comput. Sci.* Springer-Verlag, June 1991.

[68] H. Hering. Air traffic freeway system for Europe. Report EEC Note No. 20/05, EUROCONTROL Experimental Centre, 2005. www.eurocontrol.int.

[69] J. Hershberger and J. Snoeyink. Computing minimum length paths of a given homotopy class. *Comput. Geom. Theory Appl.*, 4:63–98, 1994.

[70] J. Hershberger and S. Suri. An optimal algorithm for Euclidean shortest paths in the plane. *SIAM J. Comp.*, 28:2215–2256, 1999.

[71] S. Hirsch and E. Leiserowitz. Exact construction of minkowski sums of polygons and a disc with application to motion planning. Technical report ECG-TR181205-01, Tel-Aviv University, 2002.

[72] H. v. d. Holst and J. C. d. Pina. Length-bounded disjoint paths in planar graphs. *Discr. Appl. Math.*, 120(1-3):251–261, 2002.

[73] P. Horák and L. Stacho. A lower bound on the number of hamiltonian cycles. *Discrete Mathematics*, 222(1-3):275–280, 2000.

[74] C.-P. Hsu. General river routing algorithm. In *Proceedings of the twentieth design automation conference on Design automation*, pages 578–583, 1983.

[75] T. Hu, A. Kahng, and G. Robins. Solution of the discrete plateau problem. *Proc. Natl. Acad. Sci.*, pages 9235–9236, October 1992.

[76] T. Hu, A. Kahng, and G. Robins. Optimal robust path planning in general environments. *IEEE Transactions on Robotics and Automation*, 9:775–784, 1993.

[77] T. C. Hu. *Integer programming and network flows.* Addison-Wesley, Reading, MA, 1969.

[78] M. Iri. *Survey of mathematical programming.* (A. Prékopa, Ed.) North-Holland, Amsterdam, Netherlands, 1979.

[79] A. Itai, C. H. Papadimitriou, and J. L. Szwarcfiter. Hamilton paths in grid graphs. *SIAM J. Comput.*, 11:676–686, 1982.

[80] D. S. Johnson and C. H. Papadimitriou. Computational complexity and the traveling salesman problem. In E. Lawler, J. Lenstra, A. R. Kan, and D. Shmoys, editors, *The Traveling Salesman Problem*, pages 68–74. John Wiley & Sons, New York, 1985.

[81] K. Kedem, R. Livne, J. Pach, and M. Sharir. On the union of Jordan regions and collision-free translational motion amidst polygonal obstacles. *Discrete Comput. Geom.*, 1:59–71, 1986.

[82] R. Kohn and G. Strang. Optimal design and relaxation of variational problems. *Communications on Pure and Appl. Math.*, (39):113–137, 1986. (Part I), 139–182 (Part II), 353–377 (Part III).

[83] R. Kohn and G. Strang. The constrained least gradient problem. In R. Knops and A. Lacey, editors, *Nonclassical Continuum Mechanics*, pages 226–243. Cambridge Univ. Press, 1987.

[84] J. Krozel, S. Penny, J. Prete, and J. S. B. Mitchell. Comparison of algorithms for synthesizing weather avoidance routes in transition airspace. In *AIAA Guidance, Navigation and Control Conf.*, Aug. 2004.

[85] J. Krozel, S. Penny, J. Prete, and J. S. B. Mitchell. Weather avoidance in transition airspace. *Journal of Guidance, Control, and Dynamics*, page to appear, 2006.

[86] Y. Kusakari, H. Suzuki, and T. Nishizeki. Finding a shortest pair of paths on the plane with obstacles and crossing areas. In J. S. et al., editor, *Algorithms and Computation*, pages 42–51. Springer, Berlin,, 1995.

[87] D. T. Lee, C. D. Yang, and C. K. Wong. Rectilinear paths among rectilinear obstacles. *Discrete Appl. Math.*, 70:185–215, 1996.

[88] C. E. Leiserson and F. M. Maley. Algorithms for routing and testing routability of planar VLSI layouts. In *Proc. 17th Annu. ACM Sympos. Theory Comput.*, pages 69–78, 1985.

[89] Y.-H. Liu and S. Arimoto. Finding the shortest path of a disc among polygonal obstacles using a radius-independent graph. *IEEE Trans. Robot. Autom.*, 11(5):682–691, Oct. 1995.

[90] M. S. Lobo, L. Vandenberghe, S. Boyd, and H. Lebret. Applications of second-order cone programming. *Linear Algebra and Applications*, 284:193–228, 1998.

[91] F. M. Maley. *Single-Layer Wire Routing and Compaction.* MIT Press, Cambridge, MA, 1990.

[92] E. A. Melissaratos and D. L. Souvaine. On solving geometric optimization problems using shortest paths. In *Proc. 6th Annu. ACM Sympos. Comput. Geom.*, pages 350–359, 1990.

[93] P. Mirchandani and R. Francis. *Discrete Location Theory.* Wiley, New York, 1990.

[94] J. S. B. Mitchell. *Planning shortest paths.* Ph.D. thesis, Stanford Univ., Stanford, CA, 1986.

[95] J. S. B. Mitchell. On maximum flows in polyhedral domains. *J. Comput. Syst. Sci.*, 40:88–123, 1990.

[96] J. S. B. Mitchell. L_1 shortest paths among polygonal obstacles in the plane. *Algorithmica*, 8:55–88, 1992.

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

[98] J. S. B. Mitchell. Shortest paths and networks. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry (2nd Edition)*, chapter 27, pages 607–641. Chapman & Hall/CRC, Boca Raton, FL, 2004.

[99] J. S. B. Mitchell and C. H. Papadimitriou. The weighted region problem: finding shortest paths through a weighted planar subdivision. *J. ACM*, 38:18–73, 1991.

[100] J. S. B. Mitchell, C. Piatko, and E. M. Arkin. Computing a shortest k-link path in a polygon. In *Proc. 33rd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 573–582, 1992.

[101] J. S. B. Mitchell, V. Polishchuk, and J. Krozel. Airspace throughput analysis considering stochastic weather. In *AIAA Guidance, Navigation, and Control Conference*, Aug 2006.

[102] M. Ohsaki and Y. Kanno. Form-finding of cable domes with specified stresses. Tech. Rep., Dept of Arch. and Arch. Syst., Kyoto Univ., Sakyo, Kyoto 606-8501, Japan, 2003.

[103] A. Orda and R. Rom. On continuous network flows. *Operations Research Letters*, 17, February 1995.

[104] J. Orlin. A faster strongly polynomial minimum cost flow algorithm. In *STOC '88: Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 377–387, New York, NY, USA, 1988. ACM Press.

[105] C. H. Papadimitriou and U. V. Vazirani. On two geometric problems related to the Traveling Salesman Problem. *J. Algorithms*, 5:231–246, 1984.

[106] E. Papadopoulou. Personal communication.

[107] E. Papadopoulou. k-pairs non-crossing shortest paths in a simple polygon. *Int. J. Comp. Geom. Appl.*, 9(6):533–552, 1999.

[108] C. D. Piatko. *Geometric Bicriteria Optimal Path Problems.* Ph.D. thesis, Cornell University, 1993.

[109] J. Plesnik. The NP-completness of the Hamilton cycle problem for planar digraphs of degree bound two. *Inform. Process. Lett.*, 8(4):199–201, 1979.

[110] J. Prete and J. S. B. Mitchell. Safe routing of multiple aircraft flows in the presence of time-varying weather data. In *AIAA Guidance, Navigation and Control Conf.*, Aug. 2004.

[111] J. Reif and H. Wang. The complexity of the two dimensional curvature-constrained shortest-path problem. In *Proc. 3rd Workshop Algorithmic Found. Robot.*, pages 49–57, 1998.

[112] J. H. Reif and J. A. Storer. A single-exponential upper bound for finding shortest paths in three dimensions. *J. ACM*, 41(5):1013–1019, 1994.

[113] S. Sabey. On the complexity of SOKOBAN. Unpiblished, 1996.

[114] M. Sato, J. Sakanaka, and T. Ohtsuki. A fast line-search method based on a tile plane. In *IEEE International Symposium on Circuits and Systems*, pages 588–591, May 1987.

[115] M. Sharir and A. Schorr. On shortest paths in polyhedral spaces. *SIAM J. Comput.*, 15:193–215, 1986.

[116] P. Sternberg, G. Williams;, and W. P. Ziemer. The constrained least gradient problem in R^n. *Transactions of the American Mathematical Society*, 339(1):403–432, September 1993.

[117] G. Strang. Maximal flow through a domain. *Math. Program.*, 26:123–143, 1983.

[118] J. F. Sturm. Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optimization Methods and Software*, 11-12:625–653, 1999.

[119] J. Takahashi, H. Suzuki, and T. Nishizeki. Finding shortest non-crossing rectilinear paths in plane regions. In *ISAAC*, pages 98–107, 1993.

[120] A. G. Thomason. Hamiltonian cycles and uniquely edge colorable graphs. *Ann. Discrete Math.*, 3:259–268, 1978.

[121] C. Thomassen. On the number of hamiltonian cycles in bipartite graphs. *Combinatorics, Probability & Computing*, 5:437–442, 1996.

[122] C. Umans and W. Lenhart. Hamiltonian cycles in solid grid graphs. In *Proc. 38th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 496–507, 1997.

[123] R. Wein, J. P. van den Berg, and D. Halperin. The visibility-voronoi complex and its applications. In *Symposium on Computational Geometry*, pages 63–72, 2005.

[124] G. Wesolowsky. The Weber problem: History and perspectives. *Location Science*, 1:5–23, 1993.

[125] P. Widmayer, Y. F. Wu, and C. K. Wong. On some distance problems in fixed orientations. *SIAM J. Comput.*, 16(4):728–746, 1987.

[126] H. Wolkowicz. Semidefinite programming. In P. M. Pardalos and M. G. C. Resende, editors, *Handbook of Applied Optimization.* Oxford University Press, 2001.

[127] C. D. Yang, D. T. Lee, and C. K. Wong. On bends and lengths of rectilinear paths: a graph theoretic approach. *Internat. J. Comput. Geom. Appl.*, 2(1):61–74, 1992.

[128] C. D. Yang, D. T. Lee, and C. K. Wong. Rectilinear paths problems among rectilinear obstacles revisited. *SIAM J. Comput.*, 24:457–472, 1995.

[129] A. Yousefi, G. L. Donohue, and L. Sherry. High-volume tube-shape sectors (HTS): A network of high capacity ribbons connecting congested city pairs. In *Proc. 23rd Digital Avionics Systems Conference*, volume 1, pages 3.1–7, Salt-lake City, 2004.

A Appendix: Proof of Lemma 3.2

Lemma 3.2. $\pi_2^* \cap \tau = \emptyset$.

Proof. We begin by restating some known results about the structure of bdP^1, $bd(B)^2$ and $bd(T)^2$.

Proposition A.1. *bdP^1 may be decomposed into alternating sequence of straight line segments and circular arcs of radius (curvature) 1.*

Proof. bdP^1 is the locus of points which are at distance exactly 1 from (a point of) the boundary of P. Consider a short contiguous subset S of bdP^1. Since distance is a continuous function, S may be taken short enough to have all its points be at distance 1 from *the same* feature (edge or vertex) of P. If all the points in S are at distance 1 from the same point v (vertex) of P, then S is a circular arc centered at v. If different points in S are at distance 1 from different points in P, then the latter points lie on the same edge e of P and thus S is a straight line segment parallel to e.

By taking the maximal subsets of bdP^1, which are at distance 1 from the same feature of P, we obtain the decomposition of bdP^1 as claimed. □

Lemma A.2. *$bd(B)^2$ and $bd(T)^2$ each may be decomposed into alternating sequence of straight line segments and circular arcs of radius 3.*

Proof. Similar to the proof of Proposition A.1. □

To prove Lemma 3.2, we will consider the parts of τ induced by different parts of π_1^* one by one.

Proposition A.3. *$s_2 \notin \tau$.*

Proof. By construction of $(B)^2$, s_2 is "padded" by a layer of $(B)^2$ of thickness 2. Thus, π_1^* is kept away from s_2 by at least 2. $\square$

Similarly,

Proposition A.4. *$t_2 \notin \tau$.*

Thus, π_2^* "enters" and "exits" τ the same number of times. Since both s_2 and t_2 lie above π_1^*, the entry and exit points of interest lie on $bd\tau$. Let r_1 and r_2 be the *first* points of entry and exit. We will replace the subpath of π_2^* between r_1 and r_2 by the part of $bd\tau$ and show that the new path is shorter (the new path will still be feasible since it goes by the *boundary* of τ).

By Lemma A.1 the boundary of P^1 consists of straight line segments and circular arcs of radius 1. By Lemma A.2 the boundary of $(T)^2$ consists of straight line segments and circular arcs of radius 3. We may think of π_1^* as of a string "pulled taught" against P^1 and $(T)^2$. This means that π_1^* may be decomposed into a sequence of subpaths, each of which either follows the boundary of P^1 or $(T)^2$, or is a straight line segment bi-tangent to reflex portions of P^1 and $(T)^2$. Consequently, π_1^* may be decomposed into a sequence of alternating maximal straight line segments and circular arcs of radii 1 and 3; the straight line segments are bi-tangent to the consecutive arcs in the sequence. Finally, the boundary of τ will consist of circular arcs of radius 1 following $P^1(s_1, s_2)$, circular arcs of radius 3 following B^2 and straight line segments connecting the arcs. To prove $\pi_2^* \cap \tau = \emptyset$ we consider different portions of $bd\tau$ one by one.

The part of $bd\tau$ induced by the part of π_1^* running along $P^1(s_1, s_2)$ is of no interest to us. Indeed, crossing such a part of $bd\tau$ by π_2^* cannot be done without first crossing $bd\tau$ somewhere else, closer to t_1, see Figure 103.

Consider the part of $bd\tau$ induced by the part of π_1^* running along B (Fig. 104). The parts of π_1^* following bdP^1 are "padded" by a layer of $(B)^2$ of thickness 2 and thus π_2^* cannot intersect $bd\tau$ there at all. Let now ab ($a, b \in (B)^2$) be the segment of $bd\tau$ to which r_1 belongs. Let a^* (resp., b^*) be a projection of a (resp., b) on P^1. Since $b^* - P^1(b^*, a^*) - a^* - a - b - b^*$ is a closed curve, the only way for π_2^* to come back onto $bd\tau$ is through the segment ab again, i.e., $r_2 \in ab$. Thus, replacing the part of π_2^* from r_1 to r_2 by the segment $r_1 r_2$ only makes the path shorter.

Finally, consider the part of $bd\tau$ induced by a portion of π_1^* running along $(T)^2$ together with segments from and to P^1 (Fig. 105).

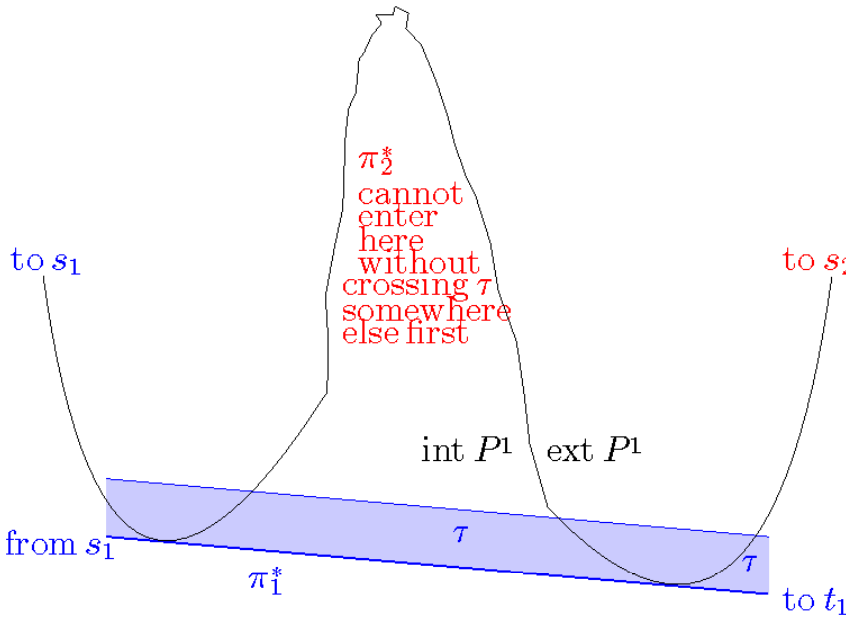


Figure 103: Part of $bd\tau$ near $P^1(s_1, s_2)$.

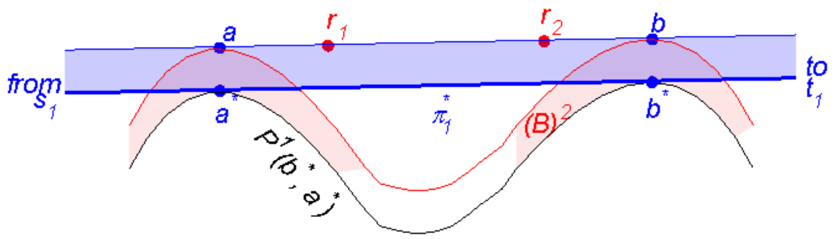


Figure 104: Part of $bd\tau$ near $(B)^2$.

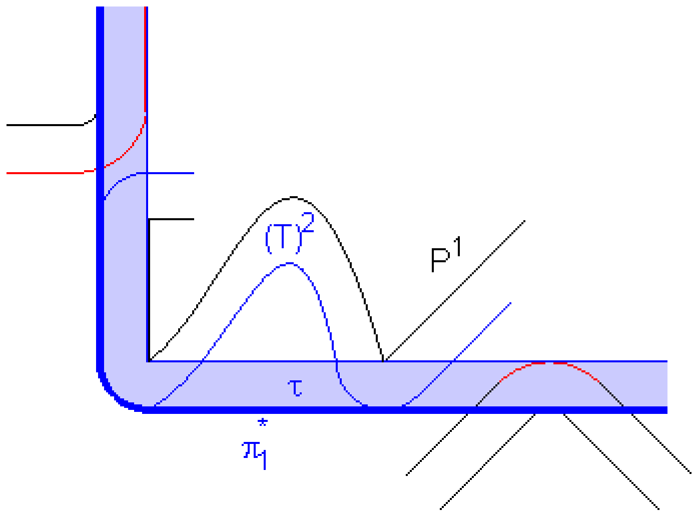


Figure 105: τ near $(T)^2$.

Observe that π_2^* certainly does not intersect π_1^*. Otherwise the part of π_2^* between consecutive points of intersection with π_2^* could have been replaced by the corresponding part of π_1^*. Indeed, there is no obstacle for π_1^* inside the closed curve $\pi_1^* - P^1(t_1, s_1)$ (only P^1 itself but it is also an obstacle for π_2^*), and π_1^* is the *shortest* path. Thus, since π_2^* is "pulled taught" against P^1 and $(B)^2$, it may enter τ only to reach a point of bdP^1 or $bd(T)^2$ inside τ.[15] Let's have a closer look at such possible point of entry.

Case I: π_2^* reaches bdP^1. Suppose π_2^* enters τ with a segment cv to reach bdP^1 at point v. Let u be the vertex of P such that $v \in bd(u)^1$, i.e., u is the vertex "responsible" for creating the part of bdP^1 in question; let au and bu be the edges of P incident to u. Without loss of generality, suppose that cv is vertical, v is below c and u is to the left of l — the supporting line of cv, Figure 106. For cv to (be the first segment of π_2^* to) intersect $bd\tau$ there should exist a point $p \in \pi_1^*$ inside $\mathcal{C}$ — the quarter-circle of radius 2 centered at v.

By the local optimality conditions, after leaving bdP^1, π_2^* goes by or to

[15]In terms of the visibility graph $VG\left(P^1 \cup (T)^2\right)$ (containing the shortest paths among P^1 and $(T)^2$ as obstacles), it means that there is a valid visibility edge $uv \in VG\left(P^1 \cup (T)^2\right)$, $u, v \in P^1 \cup (T)^2$, such that $u \notin \tau$, $v \in \tau$ (see [29, 89, 123] for discussion of the visibility graphs for planning shortest paths among obstacles bounded by circular arcs and straight line segments).

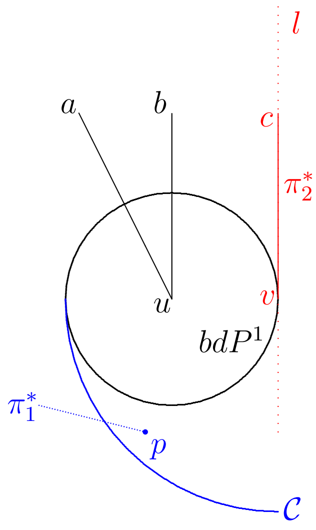


Figure 106: π_2^* reaches for bdP^1.

the left of l. For π_1^* not to intersect π_2^* there should exist an obstacle $\mathcal{O}$ inside $\mathcal{C}$, diverting π_1^* from bumping into π_2^* (Fig. 107). Consider all possible subcases:

$\mathcal{O} \subseteq P^1(s_1, s_2)$, Figure 108. As can be seen from the Figure, this can not be the case as the boundary of P^1 can not go as shown (s_1, s_2, t_2, t_1 must appear in this order when following bdP^1 clockwise).

$\mathcal{O} \subseteq (T)^2$, Figure 109. Impossible, similarly to the previous case.

$\mathcal{O} \subseteq P^1(t_1, s_1)$, Figure 110. In this case $\mathcal{O}$ is padded by a layer of B^2 of thickness 2, which serves as obstacle for π_2^*. Thus, π_2^* cannot pass by as in the Figure since $v \in \pi_2^*$ is closer than 2 to p.

Thus, it is not possible that π_2^* enters τ to reach bdP^1.

Case II: π_2^* reaches $bd(B)^2$. Let, again, cv be the vertical segment of π_2^* touching $(B)^2$ at point v (Fig. 111). For cv to (be the first segment of π_2^* to) intersect $bd\tau$ there should exist a point $p \in \pi_1^*$ inside $\mathcal{R}$ — the rectangle of width 2 with the base at cv. For π_1^* not to intersect π_2^* there

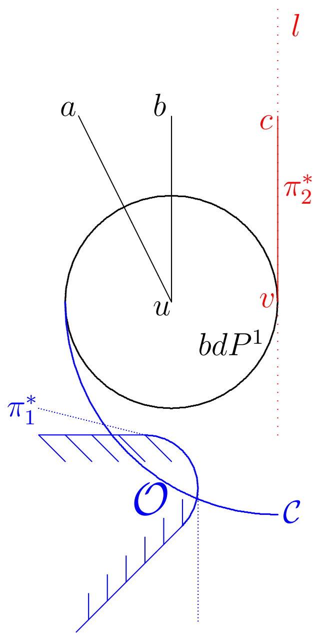


Figure 107: There must be an obstacle for π_1^* within $\mathcal{C}$.

should exist an obstacle $\mathcal{O}$ inside $\mathcal{R}$, diverting π_1^* from bumping into π_2^*. Consider all possible subcases:

- $\mathcal{O} \subseteq P^1(s_1, s_2)$, Figure 112. As can be seen from the Figure, this can not be the case as the boundary of P^1 can not go as shown (s_1, s_2, t_1, t_2 must appear in this sequence along bdP^1).
- $\mathcal{O} \subseteq (T)^2$, Figure 113. Impossible, similarly to the previous case.
- $\mathcal{O} \subseteq P^1(t_1, s_1)$, Figure 114. In this case $\mathcal{O}$ is padded by a layer of B^2 of thickness 2, which serves as obstacle for π_2^*. Thus, π_2^* cannot pass by as in the Figure since it is closer than 2 to p.

Thus, it is not possible that π_2^* enters τ to reach $bd(B)^2$. This completes the proof. □

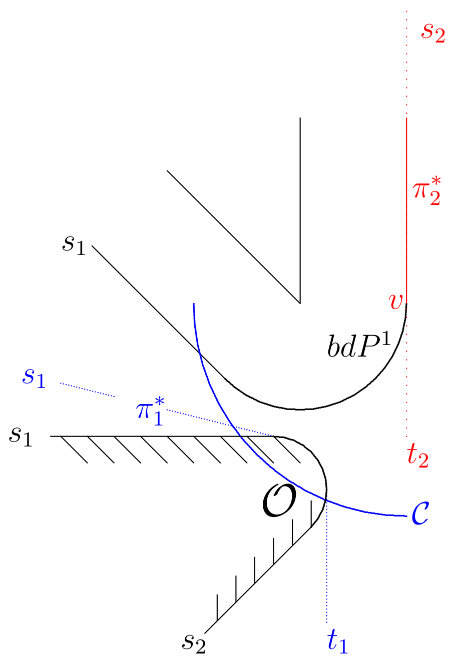


Figure 108: $\mathcal{O} \subseteq P^1(s_1, s_2)$.

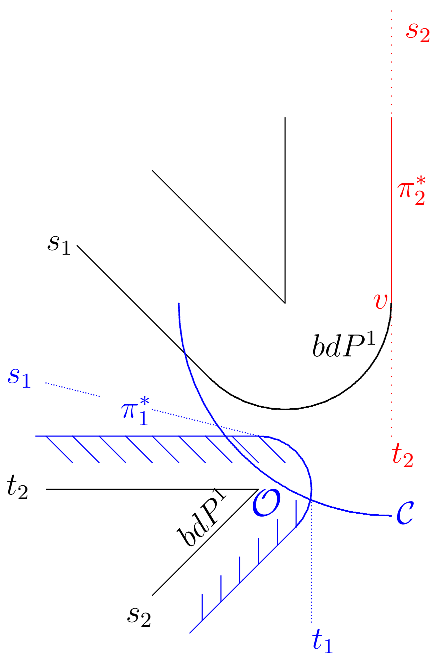


Figure 109: $\mathcal{O} \subseteq (T)^2$. The scale is not kept.

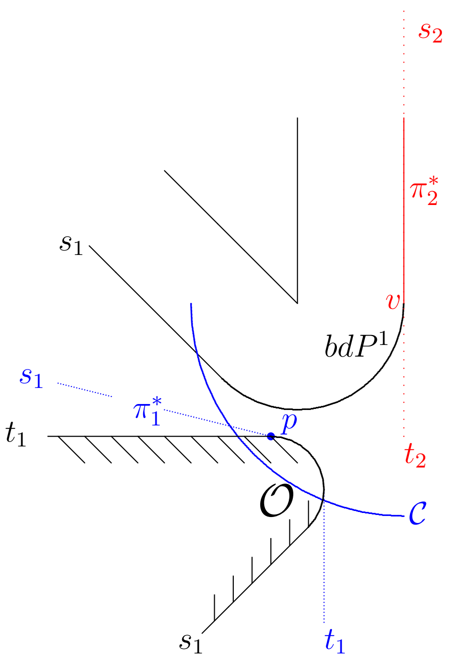


Figure 110: $\mathcal{O} \subseteq P^1(t_1, s_1)$.

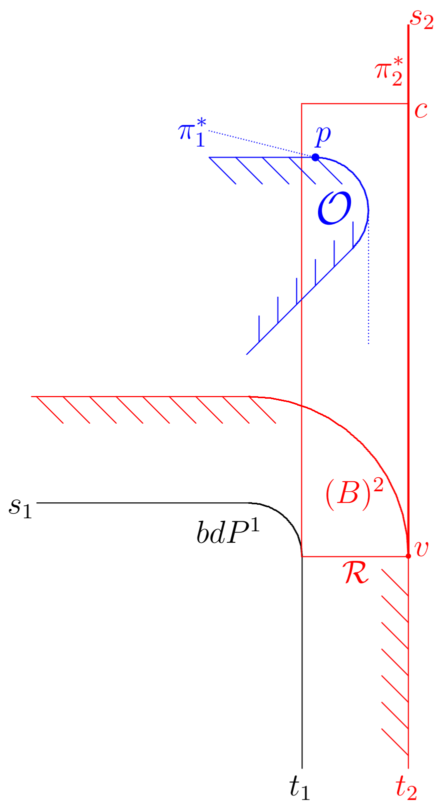


Figure 111: π_2^* reaches for $bd(B)^2$.

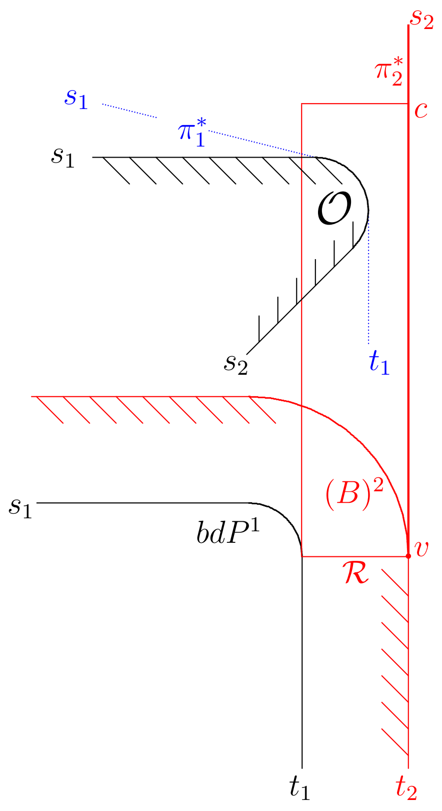


Figure 112: $\mathcal{O} \subseteq P^1(s_1, s_2)$.

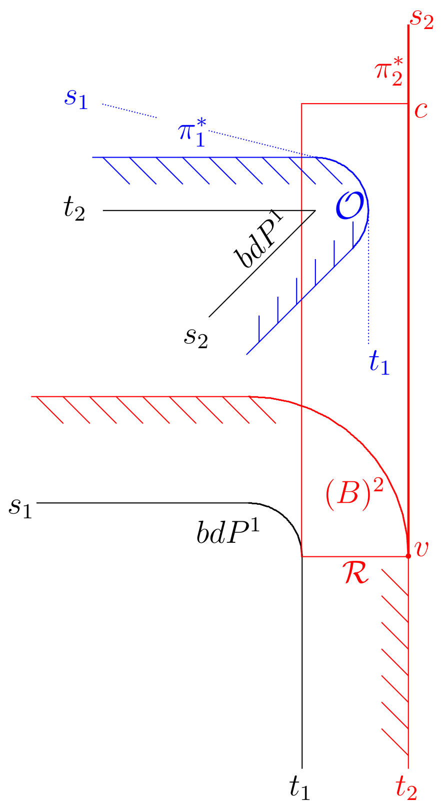


Figure 113: $\mathcal{O} \subseteq (T)^2$. The scale is not kept.

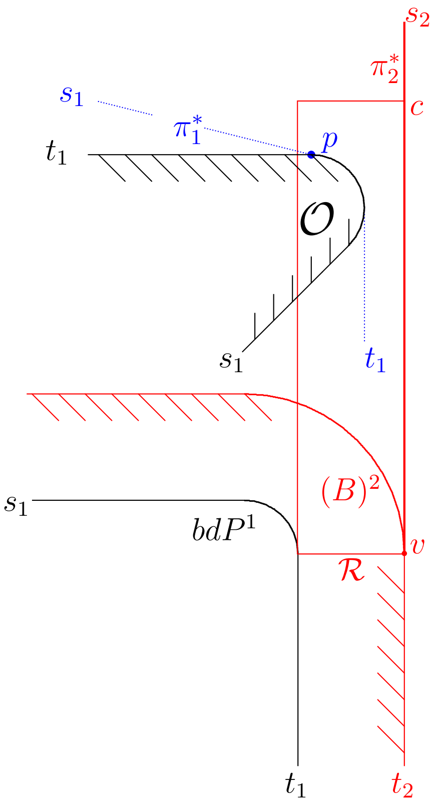


Figure 114: $\mathcal{O} \subseteq P^1(t_1, s_1)$.