Stony Brook University

Some Path Planning Algorithms in Computational Geometry and Air Traffic Management

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

Doctor of Philosophy
in
Computer Science

Stony Brook University
August 2012

Stony Brook University
The Graduate School

Shang Yang

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 – Associate Professor Jie Gao
Computer Science Department

Professor Esther M. Arkin
Department of Applied Mathematics and Statistics

Assistant Professor Jiaqiao Hu
Department of Applied Mathematics and Statistics

This dissertation is accepted by the Graduate School.

Charles Taber
Interim Dean of the Graduate School

Abstract of the Dissertation

Some Path Planning Algorithms
in Computational Geometry and Air Traffic Management

by

Shang Yang

Doctor of Philosophy
in
Computer Science

Stony Brook University
2012

Computing optimal routes subject to geometric constraints is a fundamental area of research in computational geometry. This thesis is devoted to the study of some specific geometric routing and optimization problems.

The first main area of the thesis addresses a class of multicommodity flow problems in geometric domains: For a given planar domain P populated with obstacles of different types, we consider routing thick paths corresponding to motion of vehicles of different classes, from a source edge on the boundary of P to a sink edge on the boundary of P. Each class of vehicle has an associated path width required and a given set of types of obstacles it must avoid, while it can freely pass through other types of obstacles. The problem arises in air traffic management with different classes of aircraft that must avoid various types of weather disturbances. We show that the decision problem is NP-hard even when there are only two classes of vehicles and two types of obstacles. We present approximation algorithms for the multicriteria optimization problems that arise when trying to maximize the number of routable paths. We also give heuristics and provide an experimental analysis of their effectiveness.

The second problem we address is that of computing a path or cycle that is constrained to be convex and to intersect a given set of connected, compact sets. In particular, we resolve an open problem posed more than two decades ago by Arik Tamir: Given a collection of compact sets, can one efficiently determine if there is a convex body whose boundary intersects every set in the collection? We prove that it is NP-hard, in general, to decide the existence of a convex traversal path, even if the input is a set of line segments in the plane. Further, we generalize our proof to show that deciding the existence of a convex surface stabbing a set of balls in three dimensions is NP-hard. On the positive side, we give a polynomial-time algorithm to find a convex transversal of a

maximum number of pairwise-disjoint segments in 2D, assuming the vertices of the transversal are restricted to a given set of points.

In the third part of the thesis, we investigate the problem of computing paths and trees within an uncertain geometric domain, in which the set of obstacles is not known deterministically, but is specified by a stochastic model. The problems arise in routing aircraft through uncertain weather systems, especially in the case of merging flows of aircraft arriving to a terminal airspace in the presence of uncertain events that impact the availability of airspace. We formalize the problem of computing a highly probably path in geometric settings and prove NP-hardness of the general problem. Further, we develop efficient algorithms for computing paths and trees in the presence of uncertain geometric obstacles. We apply our methods to the problem of computing routes and trees for flows of aircraft that are designed to be robust to off-nominal events that may impact the super-dense operations airspace through which they are routed.

Contents

List of Figures

Acknowledgements

My deepest gratitude goes to my advisor, Professor Joseph S. B. Mitchell, also a great mentor and friend. Joe's computational geometry course introduced me to this fascinating field. Given that I have been a dedicated fan of geometric puzzles since the age of 4, being able to work in this field is really nothing short of a dream come true. Since joining the group in the end of 2007, I have witnessed Joe's expertise in the field and his attitude and passion of doing research and I am extremely impressed. For me, it's a great lesson and an eye-opening experience. Hope some day in the future, I will love my job as much as Joe loves computational geometry. I will certainly miss the research environment he created, which is as good as one could ever imagine. There is always a perfect life-work balance; There is always enough flexibility to choose personally to work on interesting problems from the weekly Computational Geometry Seminar; And, there is no push, but always warm encouragement. To be completely honest, my 6-year PhD study working with Joe has no pain, only innumerable gains.

My sincere thanks to Professor Esther M. Arkin, one of the smartest persons I have ever met. Estie is the first person that I consult whenever I have difficulties in hardness proofs and other theoretical research problems. And, she is one of the best instructors, always very patient to explain clearly the detailed thinking process of conquering hard problems. I also learnt from her and her lectures the way to become a good presenter, including how to choose appropriate topics and how to make the audience understand.

It is a great pleasure to work with Professor Jie Gao, who is always very friendly, knowledgable and helpful. I enjoyed her lectures very much and I really like her handwritings. Special thanks to Professor Xianfeng David Gu. David was the first person I met in this country and he helped me in so many ways. Chats with David are always highly enjoyable in that he always tells many interesting stories with knowledge from various subjects.

Dr. Valentin Polishuchuk deserves special mention too. I enjoyed every discussion with him and learned from him how to write research papers properly. He passes me the confidence that a non-native English speaker can also write beautiful papers. How can I forget to thank Professor Jiaye Wang from Shandong University and Professor Wenping Wang from the University of Hong Kong? They introduced me to my first research project in 2005.

Working on NextGen projects, I have been fortunate to know Dr. Rafal Kicinger and Dr. Jimmy Krozel from Metron Aviation. The same thanks also goes to Dr. Jit-Tat Chen and Dr. Moein Ganjin. I will certainly miss our

weekly teleconferences very much.

Many thanks to Stony Brook professors that taught me and helped me in the past 6 years: Professor Leo Bachmair, Professor Michael Bender, Professor Himanshu Gupta, Professor Ker-I Ko, Professor Jerome Liang, Professor Dimitris Samaras, Professor Steve Skiena, Professor Scott Smolka, Professor Anita Wasilewska, and Professor Rong Zhao.

To my fellow students working in Joe's group, the discussions with you will forever be the sweetest memories in my life. I want to say thank you to Michael Biro, Justin Iwerks, Joondong Kim, Irina Kostitsyna, Girishkumar Sabhnani, and Jason Zou. The thanks also goes to Yao Chen, Wei Hu, Zhitao Li, Senlin Liang, Wei Xu, Yun Zeng and Dengpan Zhou. We have been enjoying our PhD life together!

In the summer of 2009, I attended the 6-day computational geometry summer school in Kanazawa, Japan. We, 25 students from all around the world, hung out together in a completely isolated seminar house (on top of a mountain, no phone, no internet, no TV...), an one-of-a-kind experience. I'd like to thank all my fellow students and the 5 professors: Tetsuo Asano, Sergey Bereg, Stefan Langerman, Ryuhei Uehara and Jack Snoeyink. More thanks go to Jack who was the session chair that helped make my research presentation debut a pleasant experience.

It's always hard to feel like at home for a student studying in a foreign country. But fortunately there are no worries for me, because of the endless love from my girlfriend Jieruo Liu. The thesis could never be completed without the sweet conversations and countless surprises from her. She is neither an expert in algorithms nor a patient person, but she listens to me carefully and patiently every time I talked to her about my research problems and every time I practiced my presentations. Actually when listening, she also discussed with me. Proved many times, her intuitive algorithmic ideas and technical support are surprisingly helpful! I don't know how to express the tons of thanks that I owe her. Besides thanks, I want to say sorry for the numerous times that she stayed up late for me when I work for various deadlines. Fortunately she is also a PhD candidate, in Economics, so I should have enough opportunities to do more than what she does for me for her. Special thanks should be given to the delicious dinners she cooked and the various best-tasting cookies and cakes she baked. Her love is always the source of my confidence and will always help me conquer difficulties.

The thesis is dedicated to my parents. Your love and support is and will always be the most precious gift in my life. Thank you for being with me.

Chapter 1

Introduction

In computational geometry, path planning is one of the most studied subjects, dealing with motions of objects in 2D or 3D geometric environments containing various types of obstacles. The goal is to determine appropriate paths for moving objects so that they satisfy various constraints (e.g., avoiding obstacles) and optimize some specified objective function. Path planning research has broad areas of applications in robotics, automation, transportation, logistics, and modern computer games [116]. In this thesis, we study some fundamental path planning problems in geometric domains, in which optimal routes are subject to geometric constraints. Specifically, we study some optimization problems arising from Air Traffic Management (ATM) in which the goal is to route thick paths (or trees) in geometric domains. In addition, we study the problem of computing a convex polygonal path (or cycle) that intersects a given set of objects.

The thesis is organized as follows.

Chapter 2 and 3 address a class of multicommodity flow optimization problems in 2D geometric domains arising in an Air Traffic Management application in which different classes of aircraft have to avoid different types of weather constraints. Specifically, given a geometric domain P, populated with different classes of polygonal obstacles, we consider the problem of routing different types of thick paths from a source edge of P to a sink edge of P. The obstacles correspond to weather disturbances while the thick paths correspond to the lanes of different kinds of aircraft with a certain width. The requirement is that each type of thick path must avoid a given set of types of obstacles, while it can pass the other types of obstacles. Chapter 2 presents mostly theoretical results, with some practical heuristics and experimental analysis. Chapter 3 discusses the problem from an empirical point of view, providing

detailed routing algorithms and experimental results on real weather data.

Chapter 4 considers an open problem proposed by Arik Tamir more than two decades ago: Given a collection of compact sets, can one efficiently determine if there a convex body whose boundary intersects every set in the collection? We present theoretical complexity analysis and algorithms for the convex transversals problem. We specifically discuss the cases in which the compact sets are line segments in 2D or balls in 3D.

Chapter 5 addresses the robust tree routing problem arising from Air Traffic Management. The goal is to merge air traffic arriving in the vicinity of an airport, while avoiding obstacles, such as hazardous weather constraints and no-fly zones. The extra challenge is to deal with the uncertainty in the weather forecast data. The chapter describes in detail the heuristic algorithms for tree generation we develop and analyze experimentally in our prototype simulation software. We also present theoretical complexity analysis for one special version of the problem, that of routing a "highly probable path" in a domain with a discrete stochastic model of uncertain obstacles.

Chapter 6 concludes with a list of open problems from the previous chapters.

Chapter 2

Routing Multi-Class Traffic Flows in the Plane[1]

In this chapter, we study a class of multicommodity flow problems in geometric domains: For a given planar domain P populated with obstacles (holes) of $K \geq 2$ *types*, compute a set of thick paths from a "source" edge of P to a "sink" edge of P for vehicles of K distinct *classes*. Each class k of vehicle has a given set, $\mathcal{O}_k$, of obstacles it must avoid and a certain width, w_k, of path it requires. The problem is to determine if it is possible to route N_k width-w_k paths for class k vehicles from source to sink, with each path avoiding the requisite set $\mathcal{O}_k$ of obstacles, and no two paths overlapping. This form of multicommodity flow in two-dimensional domains arises in computing throughput capacity for multiple classes of aircraft in an airspace impacted by different types of constraints, such as those arising from weather hazards.

[1]This work grew out of discussions during participation in the *ATM-Wx Impact Modeling Workshop 2: ATM Weather Constraint Modeling*, at the National Center for Atmospheric Research, Boulder, CO, March 3-4, 2008. We thank William Chan (NASA Ames), Nathan Downs (Metron Aviation), Jimmy Krozel (Metron Aviation), Tenny Lindholm (STAR), Joseph Prete (Metron Aviation), and Bob Sharman (STAR) for their input in formulating the algorithmic model and providing aviation expertise and weather data for our experiments. We thank Nathan Downs (Metron Aviation) for assistance in FACET simulations. FACET is provided by NASA Ames. We thank Esther Arkin, Petteri Kaski, Jukka Suomela, Girish Sabhnani and Charles Ward for helpful discussions. We thank the anonymous reviewers for their suggestions that improved the presentation of the results. This work was partially funded by NASA Ames, Metron Aviation, the National Science Foundation (CCF-0431030, CCF-0528209, CCF-0729019), and Academy of Finland (grant 138520). A preliminary version appeared in the abstracts of the 18th Annual Fall Workshop on Computational Geometry, 2008.

We give both algorithmic theory results and experimental results.

We show hardness of many versions of the problem by proving that two simple variants are NP-hard even in the case $K = 2$. If $w_1 = w_2 = 1$, then the problem is NP-hard even when $O_1 = \emptyset$. If $w_1 = 2, w_2 = 3$, then the problem is NP-hard even when $O_1 = O_2$. In contrast, the problem for a single width and a single type of obstacles is polynomially solvable.

We present approximation algorithms for the multicriteria optimization problems that arise when trying to maximize the number of routable paths. We also give a polynomial-time algorithm for the case in which the number of holes in the input domain is bounded.

Finally, we give experimental results based on an implementation of our methods and experiment with enhanced heuristics for efficient solutions in practice. Our algorithms are being utilized in simulations with NASA's Future Air traffic management Concepts Evaluation Tool (FACET). We report on experimental results based on applying our algorithms to weather-impacted airspaces, comparing heuristic strategies for searching for feasible path orderings and for computing short multi-class routes. Our results show that multi-class routes can feasibly be computed on real weather data instances on the scale required in air traffic management applications.

This chapter presents joint work with Joondong Kim, Joseph S. B. Mitchell, Valentin Polishchuk and Jingyu Zou [90].

2.1 Introduction

Many applications of path planning in polygonal domains—VLSI routing, robotics, air traffic management (ATM), sensor networks—call for finding multiple disjoint thick paths serving as "lanes" along which non-point objects may move without conflicting with each other. Studying disjoint thick paths is also of theoretical interest as it leads to developing geometric counterparts of classical network flow results: the Max-flow Min-cut, the Flow Decomposition, and Menger's theorems. In this sense, the present chapter addresses the geometric version of the *multicommodity* network flow problem.

2.1.1 Motivation

Our problem statement is natural in any multiple path routing setting involving separation standards and different constraints on paths. We suspect it has several possible applications; however, our specific motivation comes from

the domain of ATM. The aircraft differ in their capabilities, which impacts which regions of airspace they can traverse (e.g., due to hazardous weather conditions) and how much separation is needed between parallel "flows" of aircraft. In particular, one weather system can serve as an obstacle for one class of aircraft while being safely passable by another class of better equipped (or larger) aircraft. A good route planner must take this into account by possibly permitting "stronger" aircraft to fly through certain weather conditions, which serve as obstacles to "weaker" aircraft. In general, each class of aircraft is restricted to avoid certain types of airspace. See Fig. 2.1. (Note: our figures are best viewed in color.)

Our goal is to provide algorithms for decision support tools for the Next Generation Air Transportation System [162], specifically, to perform capacity estimation to determine how constraints, such as weather events, impact throughput capacity of airspace. The problem studied here is that of determining the maximum number of air lanes of multi-class aircraft that can permeate an airspace, given the constraints implied by weather forecast data. It is not expected that the routes computed with our algorithms will be the actual routes flown; many other complex issues affect the exact routes (jetways, winds, controller workload, etc). Rather, our goal is to be able to compute the maximum theoretical throughput possible across, e.g., a "flow-constrained area"

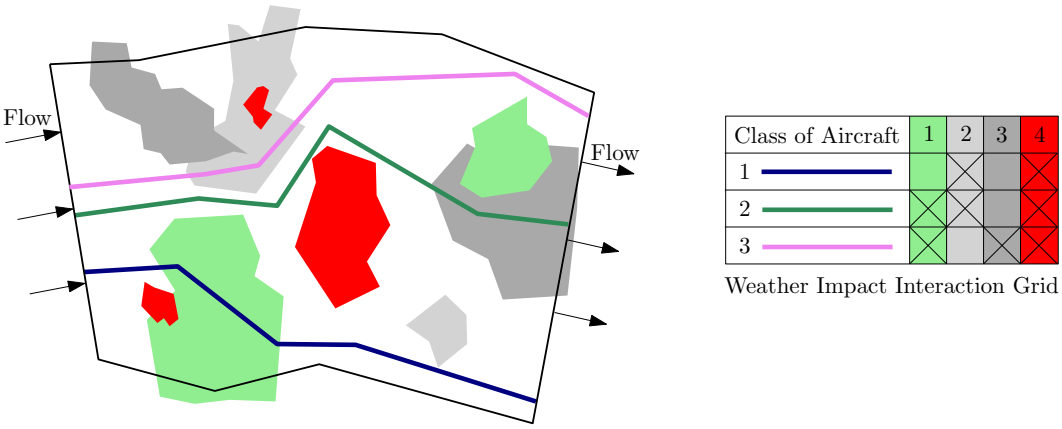


Figure 2.1: Example of the multi-class routing problem from ATM: A flow-constrained area having four different types of weather constraints impacting three different classes of aircraft. (All classes of aircraft must avoid the hard (red) constraints. As specified in the interaction grid, class 2 aircraft can freely penetrate type 3 weather events.)

(FCA), given a mixture of classes of aircraft and types of constraints, so that this information can be used as part of a decision support tool for traffic flow management.

2.1.2 Related Work

While we know of no prior results on the multi-class geometric flow routing problem studied here, multicommodity flows in discrete networks have been a subject of extensive research [35]. There is also an abundance of related work on computing multiple (single-class) paths and flows in geometric domains. A classification of existing approaches to multiple paths planning is given by van den Berg and Overmars [170]. In *prioritized* planning the paths are found one-by-one; all routed paths are declared as obstacles for a new path. The algorithms that do not use prioritized planning range from centralized over roadmap-based to decoupled (see [170] for details). A polynomial-time algorithm for finding a maximum number of thick paths in a polygonal domain is presented in [14]. The geometric versions of the Max-flow Min-cut, the Flow Decomposition, and Menger's theorems were established in [14, 127, 133, 161].

In the ATM literature, there has been considerable interest in capacity estimation using weather forecast data. Our multi-class flow problem arises when weather forecast data is translated to impact on airspace by "causality analysis" [100], which considers how different classes of aircraft respond to various types of weather hazards, such as convection, in-flight icing, turbulence, and visibility. The use of geometric flow theory for capacity estimation of a sector or an FCA in ATM is discussed in [91, 104, 108, 126, 145].

2.1.3 Summary of Contributions

1. We introduce the multi-class routing problem and prove hardness of its most basic versions: Routing two classes of vehicles among two types of obstacles, and routing paths of two distinct widths among one type of obstacle. We also prove some hardness of approximation results and the (likely) nonexistence of a fixed-parameter tractable algorithm.

2. We give approximation algorithms for multi-class routing and a polynomial-time algorithm for the case in which the number of holes in the input domain is bounded.

3. We give experimental results based on an implementation that is currently being used in simulations with NASA's Future Air traffic management

Concepts Evaluation Tool (FACET). We devise, implement, and compare efficient heuristics for exact solutions in practical settings.

2.2 Problem Formulation and Overview of the Results

The input to our problem is a polygonal domain P, consisting of an outer polygon and polygonal obstacles (holes). Let n denote the (total) number of vertices of P, and h denote the number of holes. Two edges of the outer polygon are designated as the *source* and the *sink*.

A *w-thick path* is the Minkowski sum of a usual (thin) source-sink path and disk of radius $w/2$ centered at the origin. As is common in the thick-paths literature [14, 127, 133], we assume that the polygonal domain is augmented by attaching Riemann flaps to the source and the sink so that the endpoints of the path belong to the source and the sink.[2]

The holes in the domain, as well as the paths sought are of one of K *types*. The width of a type k path is w_k, $k = 1 \dots K$. A path of type k must avoid the holes, $\mathcal{O}_k$, of type k, but may pass freely through the holes of the other types. The decision version of our problem is: Given a set of numbers $N_1, \dots, N_K$, determine if it is possible to route N_k width-w_k paths of type k from source to sink, so that no two paths overlap. This is the *Multi-Class paths problem*.

Note that the number of paths that exist in a domain may be exponential in the input size; e.g., there may exist $\Omega(M)$ width-1 paths in a $2 \times M$ rectangle, specified with $O(\log M)$ bits. By the Continuous Flow Decomposition Theorem [133], thick paths can be encoded succinctly by representing a "bundle" of paths of total thickness W by one W-thick path. Our positive results should be understood in the sense that the paths can be found in pseudopolynomial time, or that the *representations* of the paths can be found in strongly polynomial time.

2.2.1 Type Sequence and Uppermost Paths

The source and sink edges split the boundary of the outer polygon of the domain into two parts — the *top* T and *bottom* B (Fig. 2.2). We will assume

[2] Alternatively, we may consider only the *canonical parts* of the thick paths [133], which are the "rectilinear strips" of width w with opposite sides of length w residing on the source and the sink.

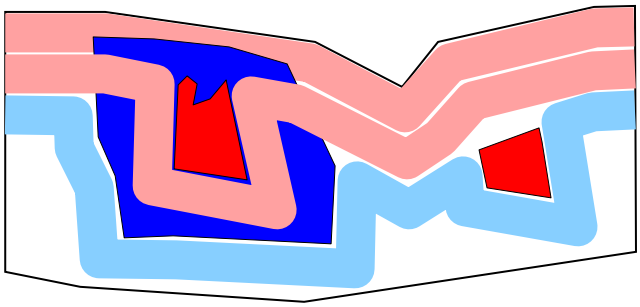


Figure 2.2: Uppermost paths for the type sequence (red, red, blue).

that in any collection of paths, the paths are numbered in the order as they are encountered when going along the source or sink from T to B. Let $\tau = (t_1, \ldots, t_M)$, $t_m \in \{1 \ldots K\}$ be a sequence of path types. We say that τ is the *type sequence* of a collection of M paths if the mth path in the collection is of type t_m.

If the type sequence of the paths sought is specified, the Multi-Class paths problem can be solved in polynomial time by routing *uppermost paths*, as we now explain. Paths $(\Pi_1, \ldots, \Pi_M)$ are called *uppermost* [14, 127] if Π_1 runs "as close as possible" to T, and Π_m runs "as close as possible" to Π_{m-1}, for $m = 2 \ldots M$. Specifically, imagine that the domain is grass, over which fire travels at speed 1; imagine also that, at the mth stage of our algorithm (when routing Π_m) the type-t_m holes, $\mathcal{O}_{t_m}$, are saturated with a highly flammable material so that whenever fire touches such a hole, the entire hole is ignited instantaneously. Ignite T at time 0, and let P' be the part of the domain that has burned by time w_1 (see [14, Theorem 2.2] for details of simulating the fire – handling events, attaching Riemann flaps, etc.). The (first) uppermost path is a w_1-thick path routed within P'. The second uppermost path is the thick path of type t_2 routed by iteratively treating the lower boundary of the first uppermost path as T and using obstacle set $\mathcal{O}_{t_2}$; the other uppermost paths are defined recursively in a similar manner.

We obtain the following straightforward extension of Theorem 2.2 in [14]:

Theorem 2.2.1. *If there exist M paths with type sequence τ, then there exist uppermost paths with type sequence τ, and a representation of the paths can be found in $O(nh+n\log n+M)$ time.*

For the case in which the holes are all of the same type, and paths only differ by their widths, we also have a "query" version of Theorem 2.2.1 (the benefit

of the query version is the running time). In contrast with Theorem 2.2.1, the algorithm for Theorem 2.2.2 does not produce the paths; it just tests whether the routing is possible. (We have been unable to obtain a result similar to Theorem 2.2.2 in the case that holes are of two or more different types; this remains open.)

Theorem 2.2.2. *If all holes are of the same type, a graph can be built in time $O(nh)$ such that given a type sequence τ, it can be tested in $O(M + h^2 \log M)$ time whether it is possible to route the M paths with type sequence τ, by solving a variation of the shortest path problem in the graph.*

See Section 2.3 for the proof of the theorem.

For most of the chapter we speak about two types of paths ($K = 2$), and concentrate on two cases:

Red/Blue paths problem: The paths have the same width ($w_1 = w_2 = 1$), but the holes are of two types, *Red* and *Blue*.

Two Widths paths problem: The holes are of one type, but the paths have different widths, $w_1 = 2, w_2 = 3$.

We focus on these special cases only for ease of presentation, and actually lose no generality by considering them: we give hardness proofs for these restricted cases, and our algorithms extend to the case of an arbitrary number of types of paths of many different widths.

Our main hardness result, proved in Section 2.4, is as follows.

Theorem 2.2.3. *(i) Given two integers, r and b, and a polygonal domain with red and blue holes, it is NP-hard to decide if it is possible to route r red and b blue pairwise-disjoint width-1 paths. (ii) Given two integers, N_1 and N_2, and a polygonal domain with (monochromatic) holes, it is NP-hard to decide whether it is possible to route N_1 width-2 and N_2 width-3 pairwise-disjoint paths.*

In fact, our hardness proof for (i) holds even when only blue obstacles are used in the reduction. Thus, Theorem 2.2.3(i) can be strengthened to show that an even more special case is hard:

Corollary 2.2.4. *Given two integers, r and b, and a polygonal domain with* only blue *holes, it is NP-hard to decide if it is possible to route r red and b blue pairwise-disjoint width-1 paths.*

Since finding an exact solution to our bicriteria optimization problem is NP-hard, we turn the attention to approximation algorithms. Suppose that there exist r red and b blue paths in P. There are (at least) three approaches to approximation of the problem:

(i) approximate the maximum number of routable blue paths while making sure that r red paths are routed;

(ii) approximate the numbers of paths of both colors, i.e., give an algorithm to route αr red and βb blue paths, for some $\alpha, \beta \in (0, 1)$; and,

(iii) approximate the maximum number of routable blue paths while making sure the total number of routed paths stays equal to $r + b$.

We leave approach (i) as an open problem. In Theorem 2.2.7 we show that (ii) is possible for essentially any α, β with $\alpha + \beta = 1$. The reduction employed in the proof of Theorem 2.2.3 shows that (iii) is NP-hard:

Corollary 2.2.5. *Let P a polygonal domain with n vertices in which there exist r red and b blue paths. Let $\gamma = \Omega(n^{1/6-\varepsilon})$ be a number, for some $\varepsilon > 0$. Unless P=NP, one cannot find in polynomial time a set of $r - \gamma b$ red and γb blue paths in the domain.*

In the Red/Blue (resp., Two Widths) paths problem, a type sequence is just a sequence of R's and B's (resp., 2's and 3's). In certain cases, one can go through all possible type sequences in polynomial time. For example, suppose that in the Red/Blue paths problem, the number of color changes in the type sequence is bounded by a number L. Then the number of different type sequences, for a total of M red and blue paths, is $O(M^{O(L)})$ – polynomial for a constant L. For each of the sequences, we can use Theorem 2.2.1 to route the red/blue paths or to conclude that it is not possible. Define a *switch* to be a change from R to B, or from B to R (resp., from 2 to 3, or from 3 to 2) in the type sequence for the Red/Blue (resp., Two Widths) paths problem. Then, we have

Lemma 2.2.6. *If the number of switches is at most L, both the Red/Blue and the Two Widths paths problems can be solved in $O(\text{poly}(n\, M^{O(L)})$ time.*

One may hope to have a fixed-parameter tractable, $O(f(L)\ \text{poly}(\text{n}, M))$-time algorithm (where f could be exponential). However, in our reduction from INDEPENDENT SET to the Red/Blue paths problem (proof of Theorem 2.2.3),

the existence of an independent set of size L in the graph implies a solution to the Red/Blue paths problem with at most $L + 1$ switches. Thus, an $O(f(L)\ \mathrm{poly(n}, M))$ algorithm for the Red/Blue paths problem would imply an $O(f'(L)\ \mathrm{poly}(n))$ algorithm for INDEPENDENT SET in an n-vertex graph, which is unlikely to exist due to $W[1]$-completeness of the problem [51]. Thus, our problem is $W[1]$-hard with respect to the number of switches.

On the positive side, we show that *any* sequence of paths can be approximated by a sequence with a small number of switches. Choosing the *best* type sequence out of all sequences with few switches, and appealing to Lemma 2.2.6, we obtain an approximation algorithm for the Red/Blue paths problem. Specifically, in Section 2.5 we prove

Theorem 2.2.7. *Let r and b, $r \leq b$, be two integers such that there exist r red and b blue thick paths in P. Let $L \leq r + b$ and $\ell \leq L$ be two arbitrary integers such that r/L and b/L are integers. One can find $\frac{\ell}{L}r$ red and $\frac{L-\ell}{L}b$ blue paths in $O(\mathrm{poly}(n, r^L))$ time.*

An analogous statement holds for the Two Widths paths problem.

As another application of Lemma 2.2.6, we give a polynomial-time algorithm for the case when h, the number of holes in the domain, is small. In this case, the number of relevant "threadings" of the paths is small, and within a subsequence of paths of common threading, it is enough to have at most one switch in the type sequence. We prove in Section 2.6:

Theorem 2.2.8. *If the number of holes in the domain is bounded ($h = O(1)$), both the Red/Blue and the Two Widths paths problems can be solved in polynomial time.*

In the remainder of the chapter we give proofs of the above theorems. In Section 2.7, we discuss an implementation and experimental results.

2.3 Testing type sequence feasibility in the monochromatic case

In this section the holes and the paths are all of the same color; the paths differ only in width (i.e., a type sequence is a sequence of numbers – the paths widths). We build a data structure to answer efficiently the following queries: "Given a type sequence τ, is it possible to route a set of paths with type sequence τ?":

Theorem 2.2.2 (stated again). *If all holes are of the same type, a graph can be built in time $O(nh)$ such that given a type sequence τ, it can be tested in $O(M + h^2 \log M)$ time whether it is possible to route the M paths with type sequence τ, by solving a variation of the shortest path problem in the graph.*

Note that here we do not find the paths themselves; we only test whether the routing is possible in principle.

Proof. We build the graph on the holes of the domain. Specifically, the *critical graph* of the domain [14, 65, 127] has a vertex for each hole, for T, and for B; the length of an edge between two vertices is equal to the Euclidean distance between the corresponding holes (Fig. 2.3). The graph can be built in $O(\mathrm{n}h)$ time by using the linear time algorithm in [10] to compute the Euclidean distance between every pair of holes. The length of a shortest T-B path in the graph is equal to the value of the maximum flow [127]. In the *thresholded* version of the graph the length of each edge is thresholded down to the nearest integer; the length of a shortest T-B path in the graph is equal to the maximum number of width-1 paths that can be routed though P. See [14, 65, 127] for details.

We use a Dijkstra-like algorithm to find a shortest T-B path in the critical graph, conforming to the sequence τ. We label the vertices of the graph by positions in τ. The *permanent* label $\ell(v)$ assigned to a vertex v after the algorithm completes, is the largest index such that the paths $1, \ldots, \ell(v)$ can be routed between T and the hole corresponding to v.

We start with assigning permanent label $\ell(T) = 0$ and *temporary* labels $\ell(\cdot) = \infty$ for the other vertices (just like in Dijkstra's algorithm, the temporary label of each vertex is an upper bound on its permanent label). Next, we propagate the labels, Dijkstra-style from the vertex v with the smallest label. For each edge (v, u) out of v, we see how far along τ it is possible to go by that edge, and update the label of u:

$$\ell(u) \quad \leftarrow \quad \min \left\{ \quad \ell(u) \quad , \quad \arg\max_{m} \sum_{i=\ell(v)+1}^{m} w_{t_i} \le d(u,v) \quad \right\}$$

where $d(u, v)$ is the length of the edge (u, v) in the critical graph, and w_{t_i} is the width of the path of type t_i. The propagation along an edge takes $O(\log M)$ time (after storing an array of τ's partial sums).

Just like in Dijkstra's algorithm, the induction on the label shows that the final label of B is the length of the longest subsequence of τ (starting from t_1) that can be routed through the domain.

□

2.4 Hardness results

In this section we show that even very simple versions of our geometric multicommodity flow problem are NP-hard. For instance, an important special case of the Red/Blue paths problem is when the holes are *nested*, i.e., when the set of the red holes is a subset of blue, modeling the situation when the capabilities of one class of vehicle is a subset of the capabilities of the other class. (Then, the red paths must avoid only the red obstacles, while the blue paths must avoid *both* red and blue.) We now show that even this special case of the Red/Blue paths problem is NP-hard; in Section 2.4.3 we show the hardness of the Two Widths paths problem.

We reduce from INDEPENDENT SET [64]. Let G be a graph with n vertices and m edges. In the independent set problem, the question is: Given an integer k, do there exist k vertices of G no two of which are connected by an edge? (In this section n denotes the number of vertices in G, and k denotes the size of the independent set.) We construct from G an instance of the Red/Blue paths problem as follows.

For each vertex of G we create a *vertex gadget* (Fig. 2.4(a)). We create a column of n aligned vertex gadgets, one on top of another, forming the *vertex*

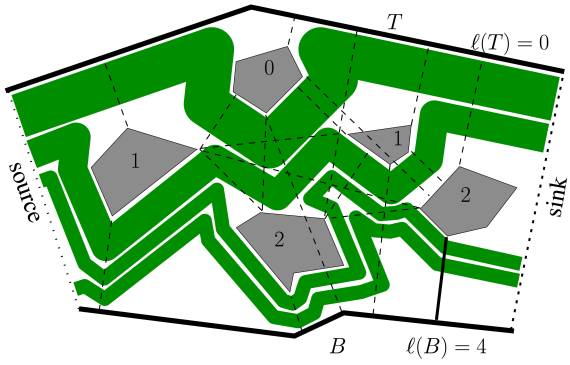


Figure 2.3: The edges of the critical graph are dashed. The numbers are labels of holes, corresponding to a sequence of widths $\tau = (3, 2, 1, 1)$. For another sequence, say, $\tau' = (3, 2, 4)$, the label of B would have been 2 since there would not be room for the third path.

part of the construction (Fig. 2.4(b)).

For each edge of G we create an *edge gadget*. To build the gadget, we first create an $8n$-by-$8n$ square with top and bottom sides being red segments; we put n equally spaced blue segments of height 4 along the right side of the square (Fig. 2.4(c)). If edge e is incident to a vertex i, we add length 1 to the top and the bottom of the ith obstacle in the edge gadget corresponding to e (thus, there are exactly two stretched obstacles in each edge gadget). Finally, the top and the bottom boundary of each edge gadget are shifted by 1 up and down (Fig. 2.4(d)).

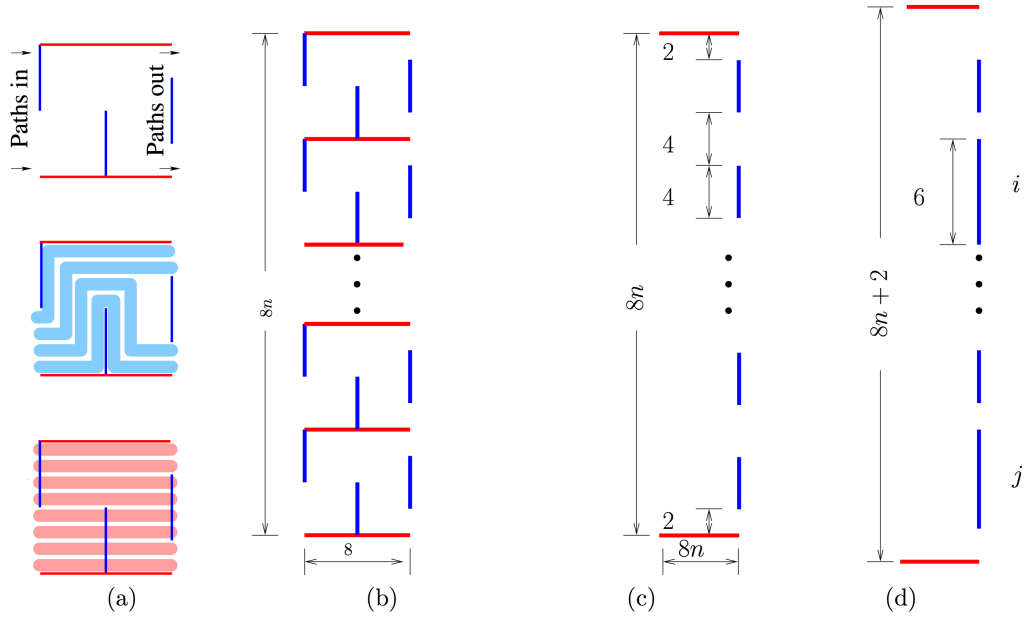


Figure 2.4: (a) Top: the vertex gadget is an 8-by-8 square with top and bottom sides being red segments; there are three blue obstacles inside the gadget, each is a vertical segment of height 4. (a) Middle and bottom: if the paths going through the gadget are of one color, they are either (at most) four blue paths or (at most) eight red paths. (b): the vertex part — n stacked vertex gadgets. (c): each edge gadget is built from an $8n$-by-$8n$ square with n blue obstacles, each being a height-4 blue segment. (d): in the gadget for an edge (i, j), we stretch ith and jth obstacles by extending them upwards and downwards by length 1; also, the top and the bottom sides of each edge gadget are shifted by 1 up and down respectively.

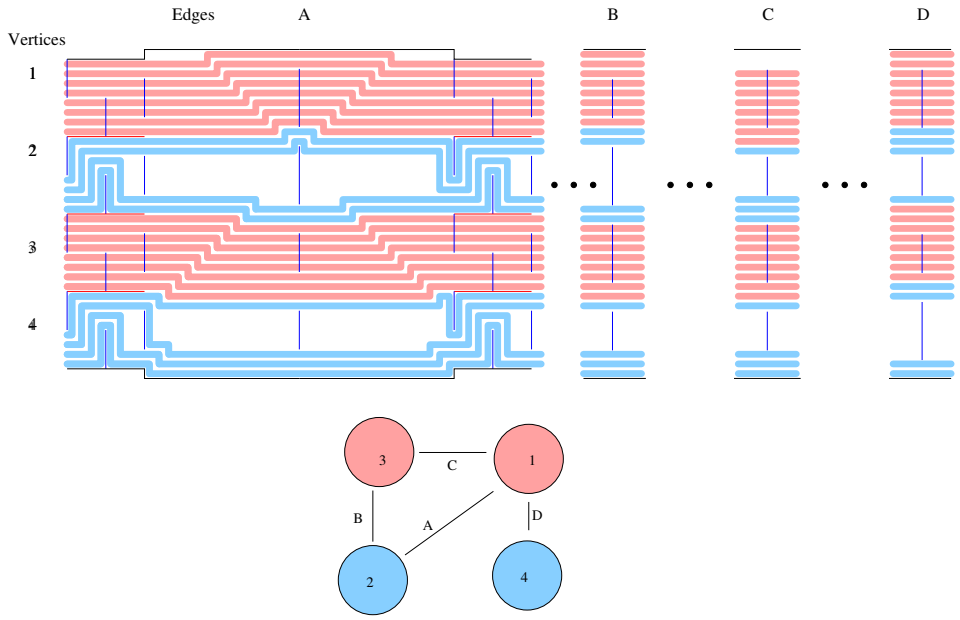


Figure 2.5: The vertex parts and the edge gadgets are put one after another. The edge gadgets are not to scale. Stretching the obstacles shifts the paths by 1 up and down; the shifted paths fit fine into the gadgets because the top and the bottom of the gadgets were shifted by 1 too. The outer polygon of the domain is shown black. This example shows the construction for the graph at the bottom.

To finish the construction, we put the vertex part and the m edge gadgets side by side from left to right; we align the obstacles in the edge gadgets with the rightmost obstacles in the vertex part. We then insert a vertex part between consecutive edge gadgets; this way, the paths are always aligned in the same way before entering any edge gadget (Fig. 2.5). Overall, our construction has, from left to right: vertex part – edge 1 gadget – vertex part – edge 2 gadget – vertex part – $\cdots$ – vertex part – edge m gadget. Since the paths are non-crossing, the ordering of the paths from top to bottom is the same in every gadget.

We now prove that there exists an independent set of size k in G if and only if $8(n-k)$ red and $4k$ blue paths can be routed all the way from the left of the construction to the right.

First, suppose there is an independent set of size k in G. Route four blue paths through each vertex gadget that corresponds to a vertex in the independent set; route eight red paths through the other gadgets. We claim

that the paths may pass through all edge gadgets. Indeed, consider any edge gadget. If the gadget did not have obstacles with additional length, the paths could go through the gadget in the same way they came out of the vertex gadgets. Adding height 1 to the top and bottom of an obstacle in the gadget may have a pair of blue paths shifted up and down, causing also shifting of the other paths. But since the blue paths go through vertex gadgets that collectively correspond to an independent set, there is at most one pair of shifted blue paths within one edge gadget. Thus, the shifted paths will fit into the extra space at the top and the bottom. This proves that if there is an independent set of size k in G, there exist $8(n-k)$ red and $4k$ blue paths in our instance of the Red/Blue paths problem.

On the other hand, suppose that there exist $8(n-k)$ red and $4k$ blue paths. We first show that no vertex gadget has both red and blue paths passing through it. Let B_0 (resp., B_1, B_2, B_3, B_4) be the set of gadgets through which 0 (resp., 1, 2, 3, 4) blue paths pass. As can be seen by inspection (Fig. 2.6), the number of red paths going through a gadget in B_0 (resp., B_1, B_2, B_3, B_4) is at most 8 (resp., 4, 2, 1, 0). Thus, the total number of red paths is at most $8|B_0|+4|B_1|+2|B_2|+|B_3| = 8n-4|B_1|-6|B_2|-7|B_3|-8|B_4| = 8n-8k-2|B_1|-2|B_2|-|B_3|$, where the first equality uses $|B_0|+|B_1|+|B_2|+|B_3|+|B_4| = n$, and the second uses $|B_1|+2|B_2|+3|B_3|+4|B_4| = 4k$. Since we assumed that there are exactly $8n-8k$ red paths, we have:

Lemma 2.4.1. $|B_1| = |B_2| = |B_3| = 0$.

By Lemma 2.4.1, there are k vertex gadgets filled with blue paths, and $n-k$ gadgets filled with red. Suppose that two gadgets, corresponding to endpoints of an edge e, are both filled with blue paths. Then the $8(n-k)$

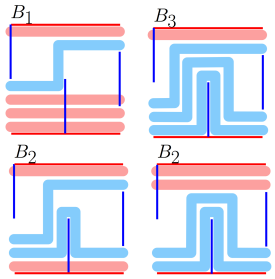


Figure 2.6: The maximum number of red paths going through a gadget with 1, 2, and 3 blue paths.

red and $4k$ blue paths will not fit through the edge gadget corresponding to e, since the topmost and the bottommost paths in the gadget will have to shift up and down by 2, and there is no space for it. Thus, the $4k$ blue paths go through vertex gadgets corresponding to an independent set of size k in G.

This proves

Theorem 2.4(i) (stated again). *Given two integers, r and b, and a polygonal domain with red and blue holes, it is NP-hard to decide if it is possible to route r red and b blue pairwise-disjoint width-1 paths.*

2.4.1 Only blue obstacles

In the above construction we can replace all red obstacles with blue obstacles. Indeed, the only place where the red obstacles appear is the vertex gadget. Lemma 2.4.1 remains true after the replacement, and so does the claim that independent sets of size k in G are in one-to-one correspondence with $8(n-k)$ red and $4k$ blue paths. Thus, the Red/Blue paths problem is hard even when restricted to instances with only blue obstacles:

Corollary 2.2.4 (stated again). *Given two integers, r and b, and a polygonal domain with* only blue *holes, it is NP-hard to decide if it is possible to route r red and b blue pairwise-disjoint width-1 paths.*

2.4.2 Hardness of approximation

Assume there exist r and b red and blue paths in the problem instance constructed from a graph G with N vertices and M edges. Let now $n = O(NM) = O(N^3)$ be the complexity of the domain constructed from G. We know from the reduction that there exists an independent set of size $b/4$ in G. If we could find $r - \gamma b$ red and γb blue paths, for some γ, we could find an independent set of size $\gamma b/4$. This is not possible in polynomial time (unless P=NP) for $\gamma = \Omega(N^{1/2-\varepsilon})$, where $\varepsilon > 0$ is an arbitrary positive number [44]. This means that one cannot approximate, to within any factor $\gamma = \Omega(n^{1/6-\varepsilon})$, the maximum number of routable blue paths while keeping the total number of routed paths equal to $r + b$:

Corollary 2.2.5 (stated again). *Let P a polygonal domain with n vertices in which there exist r red and b blue paths. Let $\gamma = \Omega(n^{1/6-\varepsilon})$ be a number, for some $\varepsilon > 0$. Unless P=NP, one cannot find in polynomial time a set of $r - \gamma b$ red and γb blue paths in the domain.*

2.4.3 Hardness of the Two Widths paths problem

The reduction showing NP-hardness of the Two Widths paths problem is very similar to the reduction used above for the Red/Blue paths problem. The vertex gadget is a 6-by-6 square with top and bottom sides being obstacles. If the paths going through the gadget are of the same thickness, they are either (at most) three width-2 path or (at most) two width-3 paths. As in the proof of hardness of the Red/Blue paths problem, n vertex gadgets are stacked one on top of another forming the *vertex part* (Fig. 2.7).

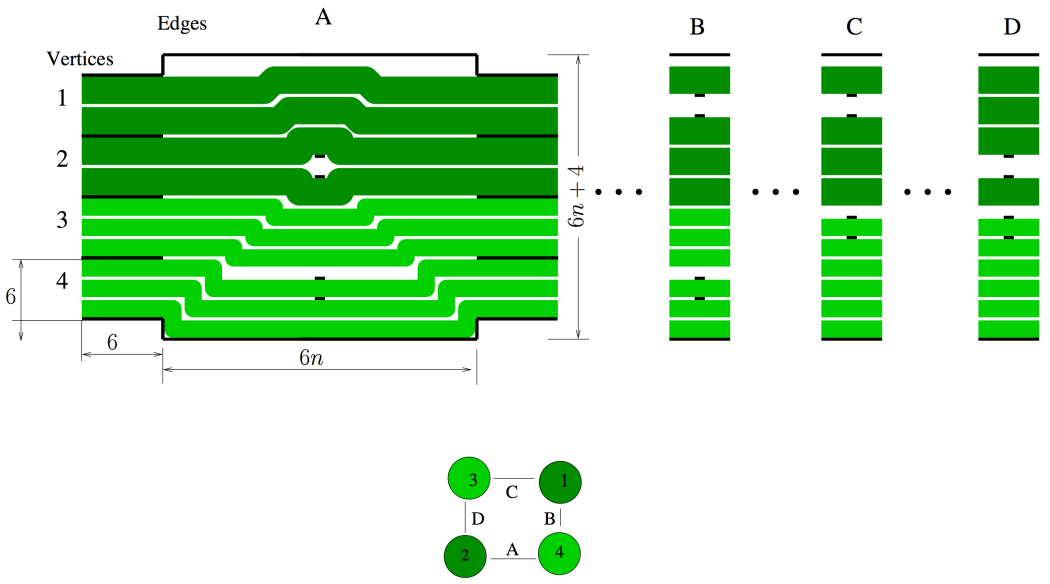


Figure 2.7: Construction for the proof of Theorem 2.2.3(ii). The edge gadgets are not to scale. This example shows the construction for the graph at the bottom.

Each edge gadget is a $6n$-by-$(6n + 4)$ rectangle with top and bottom sides being obstacles. In the gadget for an edge (i, j), we put a pair of point obstacles at distance 2 apart, aligned with vertex gadgets i and j.

The rest of the construction is identical to the one for the Red/Blue paths problem. The edge gadgets are put next to the vertex part, one-by-one from left to right. We also insert a vertex part between consecutive edge gadgets to ensure the paths are aligned in the same way before entering any edge gadget. There is an independent set of size k in G if and only if we can route $3(n - k)$ width-2 and $2k$ width-3 paths; the wider paths must go through vertex gadgets,

corresponding to an independent set.

Theorem 2.2.3(ii) (stated again). *Given two integers, N_1 and N_2, and a polygonal domain with (monochromatic) holes, it is NP-hard to decide whether it is possible to route N_1 width-2 and N_2 width-3 pairwise-disjoint paths.*

2.5 Approximation

We now turn to positive results. In this section we show how to approximate any type sequence by a sequence with few switches. Choosing the best sequence with the few switches allows one to approximate simultaneously the number of paths of each color in polynomial time:

Theorem 2.2.7 (stated again). *Let r and b, $r \leq b$, be two integers such that there exist r red and b blue thick paths in P. Let $L \leq r + b$ and $\ell \leq L$ be two arbitrary integers such that r/L and b/L are integers. One can find $\frac{\ell}{L}r$ red and $\frac{L-\ell}{L}b$ blue paths in $O(\mathrm{poly}(n, r^L))$ time.*

Proof. Consider the collection of r red and b blue paths. Consider also $L + 1$ (thin) source-sink paths $\pi_0^*, \pi_1^*, \ldots, \pi_L^*$ (where $\pi_0^* = T$, $\pi_L^* = B$) that split the domain so that there is exactly $(r + b)/L$ thick paths (both red and blue) between π_{i-1}^* and π_i^* for each $i = 1 \ldots L$ (Fig. 2.8). Call the part of the domain between π_{i-1}^* and π_i^* the ith *slot*. Let $R^* = (r_1^*, \ldots, r_L^*)$, $B^* = (b_1^*, \ldots, b_L^*)$, be the sequences of the numbers of red and blue paths in the slots; $r_1^* + \cdots + r_L^* = r$, $b_1^* + \cdots + b_L^* = b$, $r_1^* + b_1^* = \cdots = r_L^* + b_L^* = (r + b)/L$.

We go through all $O(r^L)$ possible representations of r as a sum of L integers each less than or equal to $(r + b)/L$. The integers specify one possibility for how many of the r paths reside in each of the L slots. For each representation $R = (r_1, \ldots, r_L)$, we run the algorithm RedBluePaths that routes successively either r_i red uppermost paths or $\frac{r+b}{L} - r_i$ blue uppermost paths; the choice of the color is determined by whether r_i is one of the ℓ largest numbers in R:

Algorithm RedBluePaths(R)
Input. $r, b \in \mathbb{N}$; domain in which there exist r red and b blue paths; integers L, ℓ; sequence of integers $R = (r_1, \ldots, r_L)$, such that $r_1 + \cdots + r_L = r$.
Output. A collection of $\ell r/L$ red and $(L - \ell)b/L$ blue paths (if one exists).

```
1  max_ℓ(R) ← indices of ℓ largest integers in R
2  for i = 1 to L
3  if i ∈ max_ℓ(R)
4  route r_i uppermost red paths
5      else
6  b_i ← (r+b)/L − r_i
7  route b_i uppermost blue paths
8  endif
9  endfor
```

Now, one of the representations examined will be R^*. Let Π_i^* be the set of uppermost paths routed by RedBluePaths(R^*) in the ith **for** loop; Π_i^* is a collection of either r_i^* red *or* b_i^* blue uppermost paths, depending on whether $i \in \max_\ell(R)$ or not. Since the first slot contains r_1^* red *and* b_1^* blue paths, the uppermost paths Π_1^* (which are *either* r_1^* reds *or* b_1^* blues) will stay within the slot:

Fact 1. *The lower boundary of* Π_1^* *is* above π_1^*.

The second slot contains r_2^* red *and* b_2^* blue paths. Again, r_2^* red *or* b_2^* blue *uppermost* paths, routed within the slot (i.e., treating π_1^* as the top of the domain), will stay within the slot. RedBluePaths(R^*) actually routes Π_2^* treating the lower boundary of Π_1^* as the top. Thus, by Fact 1, Π_2^* will not cross π_2^*. By induction, we obtain that RedBluePaths(R^*) will successfully route the paths without crossing $\pi_L^* = B$.

The number of red paths routed by RedBluePaths(R^*) is

$$\sum_{i \in \max_\ell(R)} r_i^* \quad \geq \quad \ell r/L \quad .$$

The number of blue paths routed is

$$\sum_{i \notin \max_\ell(R)} b_i \quad = \quad b - \sum_{i \in \max_\ell(R)} b_i \quad \geq \quad b - \ell b/L$$

□

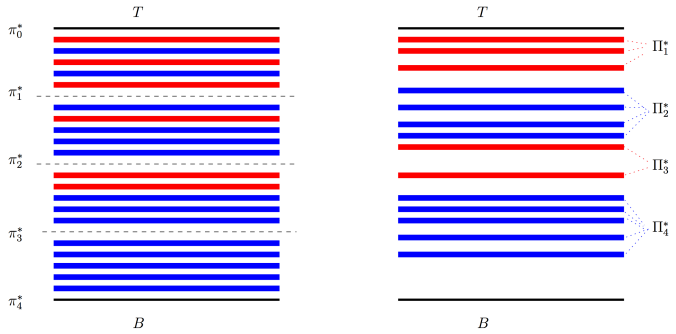


Figure 2.8: Left: In ith slot there exist r_i^* red *and* b_i^* blue paths; $r_i^* + b_i^* = (r+b)/L$. Right: RedBluePaths($\mathcal{R}^*$) routes *either* r_i^* red *or* b_i^* blue uppermost paths; thus, the paths do not go below the boundary of the slot.

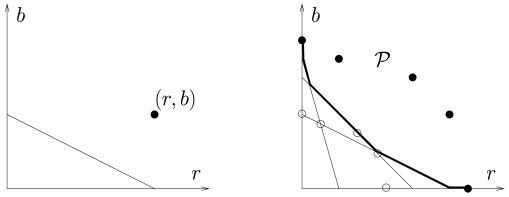


Figure 2.9: Left: The line segment $(0, b)$-$(r, 0)$ is the dual of (r, b). Right: We obtain the upper envelope of the segments dual to the Pareto optimal pairs (filled circles); the half-optimal solutions (hollow circles) are below the envelope.

Approximating Pareto-optimal solutions. Applying Theorem 2.2.7 to all possible values of (r, b) such that there may exist r red or b blue paths, we obtain an approximation to the *Pareto frontier* of optimal solutions to the problem. Specifically, for a pair of numbers (r, b) let the *dual* of the pair be a line segment from $(0, b)$ to $(r, 0)$ in the (r, b)-plane (Fig. 2.9). Let $\mathcal{P}$ be the set of *Pareto optimal* pairs (r, b), i.e., such that there exist r and b red and blue paths through the domain ((r, b) is feasible), but $(r, b+1)$ and $(r+1, b)$ are not feasible pairs. We say that pairs $(r/2, b/2)$ for $(r, b) \in \mathcal{P}$, are *half-optimal* solutions.

Let r^* (resp., b^*) be the maximum number of red (resp., blue) paths that can be routed in the domain without routing any blue (resp., red) paths. For each pair $(r, b) \in [0, r^*] \times [0, b^*]$ and each $\ell = 0, 1, \ldots, r$ we apply Theorem 2.2.7 with $L = r$. This gives a set of line segments, which includes the segments dual to the points in $\mathcal{P}$. Since for each pair $(r, b) \in \mathcal{P}$ we obtain at least

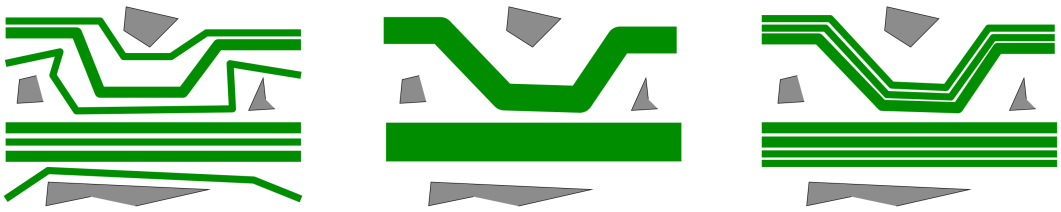


Figure 2.10: Two bundles (left) may be transformed into two thick paths (middle). Then the order of width-2 and width-3 paths may be changed so that there is one switch per bundle.

$(r/2, b/2)$ paths (by setting $\ell/L = 1/2$), half-optimal solutions lie below the upper envelope of the segments.

2.6 Small number of holes

In this section we show that our problems are tractable when the number of holes is constant:

Theorem 2.2.8 (stated again). *If the number of holes in the domain is bounded ($h = O(1)$), both the Red/Blue and the Two Widths paths problems can be solved in polynomial time.*

Proof. We first prove the theorem for the Two Widths paths problem. Take an optimal collection of paths in the Two Widths paths problem. Fix the start and the destination of each path, and consider the collection of *shortest* paths with the fixed starts and destinations. A *threading* of a source-sink path is a vector of length h, indicating for each hole whether the hole is above or below the path [133]. Call a (maximal) set of paths of the same threading a *bundle*. By the Continuous Flow Decomposition Theorem [133], each bundle can be pulled taut so that it becomes *one* thick path, with the thickness equal to the total width of the paths in the bundle (Fig. 2.10).

Number the bundles in the order as they appear along the source when going from T to B. If a hole H is above bundle i, then it is also above bundle $i+1$. Thus, the number of threadings of the bundles – and hence the number of the bundles – is at most $h+1$. Within one bundle, the paths may be reordered so that there is at most one switch in the paths' type sequence: first all width-2 paths, and then all width-3 paths.

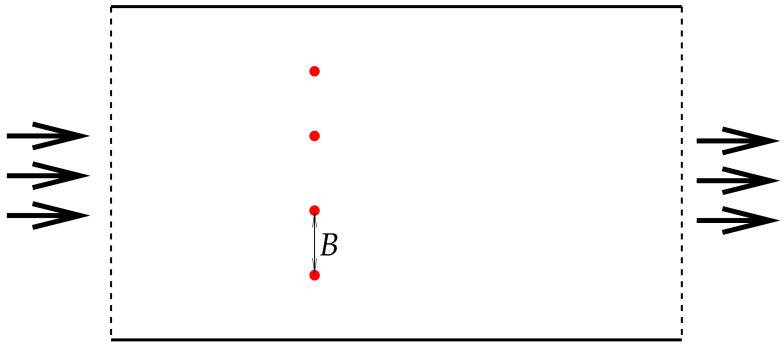


Figure 2.11: The 3-PARTITION problem asks if the numbers $a_1 \dots a_{3n}$ ($\sum a_i = nB$, $B/4 < a_i < B/2$) can be split into n groups of 3, such that the sum of numbers in each group is B. The polygonal domain created from a 3-PARTITION instance has $n-1$ holes; each hole is a point. The distance 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.

Moreover, if bundle i only has one type of paths, there is no switch in the bundle. Otherwise, the paths in the bundle can be ordered so that there is no switch when going to the bundle i from the bundle $i-1$ (e.g., if the bundle $i-1$ had width-2 paths below width-3 ones, then the bundle i can have width-2 paths above width-3 ones). This way we have at most one switch per bundle, and the total number of switches is at most $h+1$.

Theorem 2.2.8 for the Two Widths paths problem follows from Lemma 2.2.6 now. The above proof extends verbatim to the case when there are more than two, but a constant number, of path widths. On the contrary, if the number of widths is large the problem becomes NP-hard by a reduction from 3-PARTITION (Fig. 2.11).

The proof for the Red/Blue paths problem is similar. Suppose that the first path is red. So the type sequence τ starts from some number, m, of reds, and then follows some blues. Consider the first time the type sequence switches back to red; say this happens when going from kth path to $(k+1)$-st (τ_k = blue, τ_{k+1} = red). Do the "bubble sort" on the paths: Try swapping the $(k+1)$st and the kth paths; if this is possible, do the swapping (so that the kth path is red) and try swapping the kth and the $(k-1)$st path, etc. Continuing this way, either the red path floats all the way up to become the $(m+1)$st path, or it gets stuck; in the former case, try swapping the $(k+2)$nd (red) path with

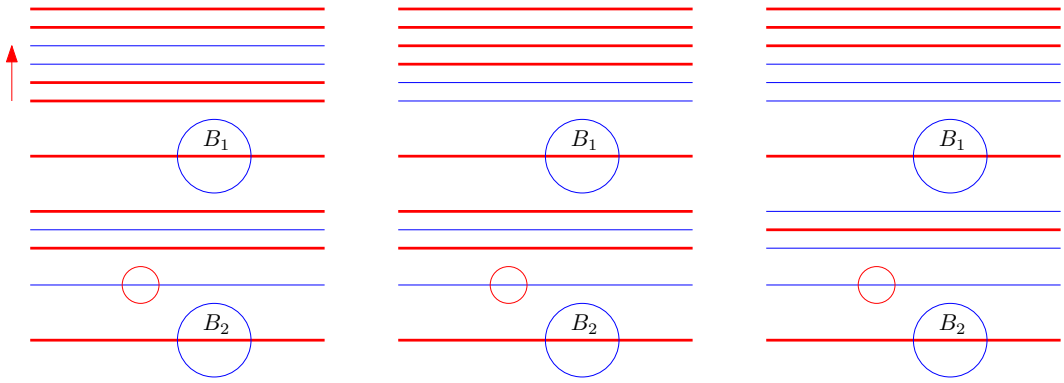


Figure 2.12: Left: Initial type sequence. Middle: Two red paths floated up, but the third is stuck because it intersects a blue hole B_1. Right: Two charged blue holes B_1, B_2 are separated by blue paths: the second layer of blue paths is below B_1 but is above B_2.

the $(k+1)$st (blue) path. In the end, either we have reduced the number of swaps by 2 (because the first layer of blue paths "drowns out" the next layer of blues), or we get stuck. Refer to Fig. 2.12.

But what does it mean to "get stuck"? It means that a red path ρ cannot be switched with a blue path β. This can only be due to either ρ passing through a blue hole B or β passing through a red hole R (or both). We charge the blue-red switch to the holes, and continue the bubble sort starting from the next blue-red switch; whenever we are stuck we again charge the blue-red switch to the holes that prevent the switch. It is easy to see that no hole is charged twice. Indeed, two consecutive charged holes are either of different colors or are separated by paths of their common color.

Thus, the number of blue-red switches is at most the number of the holes, and overall the number of the switches is at most $2h$. The claim of the theorem follows now from Lemma 2.2.6. □

2.7 Practical Heuristics: Implementation and Experiments

We have implemented algorithms for multi-class routing and applied them to weather data for use in capacity estimation experiments and in NASA's FACET simulation tool. The FACET-based scenarios are performed in collaboration with colleagues at Metron Aviation and are reported separately [102,178]; here,

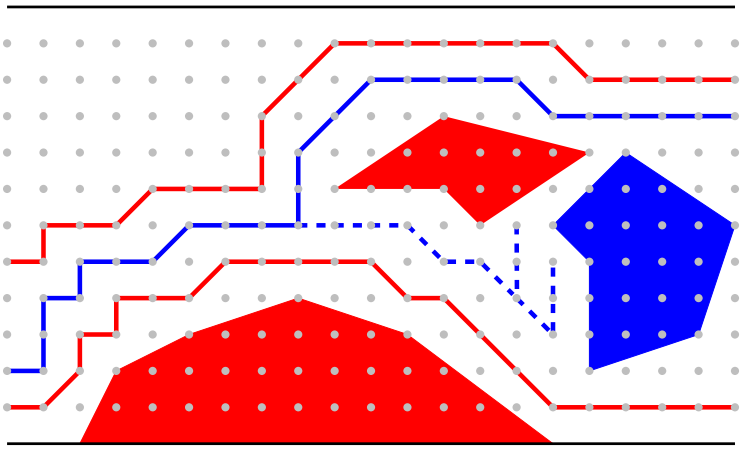


Figure 2.13: Routing bottommost paths in the grid. The grid step is equal to the path width. Starting at the lowest available grid point at the sink, the path proceeds to the next grid point, with the priority to turn to the right as much as possible; the constraints are that the path stays away from obstacles and the already routed paths, and that it is x-monotone. The routing continues until the sink is reached. If the sink is not reached, the search retracts (e.g., the DFS retracts twice when routing the second, blue path).

we report experimental results involving algorithmic design choices.

The implementation is in C++. The user interface allows one to import weather data, specify the outer boundary of P, mouse-in polygonal obstacles, and select algorithm parameters. The algorithms implemented are based on uppermost (or bottommost) filling of P with thick paths, according to a type sequence τ. Paths are inserted one by one, checking for feasibility according to the type of the path and the types of the obstacles. For purposes of these experiments, rather than doing offsetting using Voronoi methods, we use a simple method of computing bottommost routes by means of a depth-first search in a search grid that is superimposed over the domain (Fig. 2.13); this permits us to easily adapt to a wide variety of constraints, including weather data of various types, turn constraints, directionality constraints, etc. It also allows us to use sets of constraints (obstacles) that may not be disjoint from each other, as we simply have to have a predicate that tests if a given segment connecting two grid points satisfies the requisite lane width and obstacle type constraints.

For the experiments reported here, we imposed a monotonicity constraint that routes be x-monotone (in the direction of the flow of air traffic from a

western source edge to an eastern sink edge). The monotonicity constraint is often imposed on ATM-relevant routes, as one does not fly routes that are highly non-monotone, with many switchbacks.

2.7.1 Data Sets

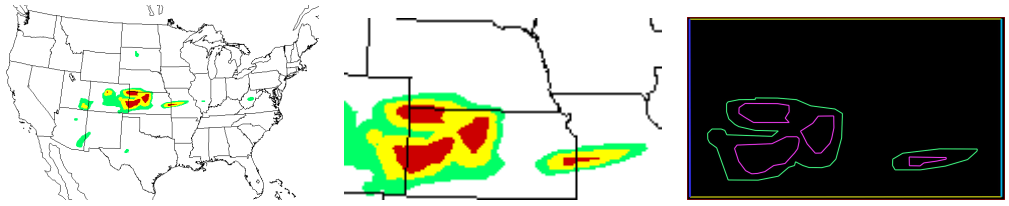
We used two kinds of data: real weather data, and simulated sets of obstacles. For real weather data, we used segmentations at two levels, a high threshold for red obstacles, and a lower threshold for blue obstacles. The weather data is of three varieties: convective weather (vertically integrated liquid), icing data, and turbulence data (Graphical Turbulence Guidance) [155]. Real weather data is usually composed of big blocks of weather constraints staying near each other, and the red constraints are typically inside the blue constraints. The example shown in Fig. 2.14 is based on a sample of convective weather data.

For simulated data, we randomly generate two sets of quadrilaterals, with one set designated as red obstacles and the other set as blue obstacles; quadrilaterals from the two sets possibly overlap with each other, and/or cross the boundary of the outer polygon. In more details, for each set, we use a uniform distribution in an axis-aligned bounding box of P to generate a pre-specified number of center points; around each center point, we first generate an L_1-metric circle of radius R from a fixed uniform distribution, and then perturb the coordinates of each vertex by $U(-R, R)$ – a random amount uniformly distributed in $(-R, R)$. An example is shown in Fig. 2.16(a).

2.7.2 Enumeration of Type Sequences

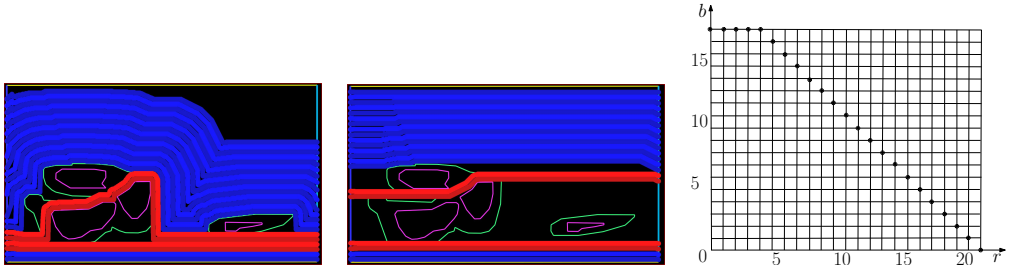
We experimented with different ways of enumerating the set of $\binom{r+b}{r}$ type sequences. If (r, b) is feasible, i.e., there exist r red and b blue paths, we hope, by choosing a smart way of enumeration, that we hit the first feasible type sequence after only a small number of steps.

Because only red paths may go through blue obstacles, intuitively it is beneficial to have red paths go through blue obstacles whenever possible in order to leave enough free space (without any obstacle) for the blue paths. Therefore, in a feasible routing scheme, it is likely that the red paths stay together inside a blue obstacle, so that the blue ones can stay together in the "free" space. This leads to an intuition that, in most cases, a type sequence with a smaller number of color changes has a greater chance of being feasible. Hence, we experimented with the following enumeration strategies:



(a) (b) (c)

Figure 2.14: Example of a real weather test case: (a). The Map of Continental United States, (b). The Region of Interest: Nebraska, Kansas and part of Wyoming, Colorado, Iowa and Missouri, (c). The test case extracted from (b): Yellow and Green hazards are light weather constraints and red ones are severe weather constraints. We consider yellow hazards to be blue constraints and red hazards to be red constraints. The test case is a typical one extracted from real weather data: the constraints are forming large clusters, staying near to each other, and the red constraints are typically inside the blue ones.



(a) (b) (c)

Figure 2.15: Example of routing in the test case from Fig 2.14(c): (a). Result of (bottommost) routing 4 red paths and 17 blue paths, (b). result of heuristic shortening, (c). the Pareto frontier of the routing instance.

(1) Starting with the initial sequence of r R's, followed by b B's, we enumerate the type sequences in lexicographically increasing order.

(2) Break into subcases according to the number, L, of color changes in the subsequence, and enumerate in one of the following ways: (a) increasing L, lex-increasing; (b) decreasing L, lex-decreasing; (c) increasing L, lex-decreasing; (d) decreasing L, lex-increasing.

If (r, b) is infeasible, the hope is to be able to conclude so *without* having to try *all* permutations. To achieve this goal, we implemented a method of pruning the search tree using red and blue *indices*: for each node of the search

(a) (b) (c)

Figure 2.16: Example of routing in a randomly generated test case: (a). The instance, (b). Result of (bottommost) routing 5 red paths and 9 blue paths, (c). result of heuristic shortening.

grid, we first compute the maximum number of red/blue paths that can pass above it. If, during our bottommost fill algorithm according to a type sequence we ever pass through a node such that its blue index is less than the number of blue paths remaining in the type sequence, we terminate early, concluding that any sequence that begins with the prefix of the sequence just tested cannot lead to a successful routing of r red and b blue paths.

2.7.3 A Heuristic for Short Paths

While it remains an open problem to compute a set of thick paths to minimize the sum (or the maximum) of the path lengths, we implemented a method for local optimization of paths for a given feasible pair (r, b). In addition to studying the length optimization problem, our goal was to compute paths that could be used in simulation experiments with FACET on real data; thus, we were expected to generate routes that were at least plausibly flyable (and bottommost fill routes fail this criterion).

Our heuristic does the following tautening of the routes computed by bottommost fill for a feasible pair (r, b) with a given type sequence. First, the topmost route (route number $r + b$, denoted γ_{r+b}) is shortened by computing a shortest thick route, γ^*_{r+b}, of the width corresponding to the class of the route, from source to sink, lying between route γ_{r+b-1} and T, the top of P, while avoiding the relevant obstacles. (Note that the homotopy type of γ^*_{r+b} may be quite different from that of γ_{r+b}.) Then, iteratively, each route γ_i is shortened by computing a shortest source-to-sink thick path that lies between γ_{i-1} and γ^*_{i+1}. Fig. 2.15(a),(b) and Fig. 2.16(b),(c) show the results of the bottommost routing and the tautening process.

2.7.4 Experimental Results

We ran each data set multiple times, for each enumeration strategy, iterating over choices of (r, b), thereby obtaining the Pareto frontier; see Fig. 2.15(c). In order to test instances that were *not* feasible, we also ran the algorithm for choices of (r, b) just above the Pareto frontier (i.e., at $(r, b^*(r)+1)$). We measure (a) the number of type sequences tested before termination, (b) whether the pair (r, b) is feasible or not, and (c) the path length (sum of lengths and max of lengths) of the solution after local tautening.

We tested over 2000 test cases on our data sets. For both feasible and infeasible pairs (r, b), we recorded the average number of type sequences tested for each of the enumeration strategies. For feasible pairs, we differentiate the test cases by *heavy loads* and *light loads*, where heavy loads test cases refer to the pairs (r, b) on the Pareto frontier and light loads cases refer to pairs (r, b) that are inside the Pareto frontier. For each of the test cases, we tested six (r, b) pairs corresponding to three light loads cases, two heavy loads cases, and one infeasible case.

The test results are shown in Table 2.1. For all strategies, we see that we obtain early termination with success far sooner than the overall average value of $\binom{r+b}{r}$. This suggests the practical efficiency of our implementation, since testing a given type sequence is a very fast operation (essentially, a depth-first search, in the grid, for the bottommost paths).

We see that for feasible pairs, though, the enumeration strategies 2(a) and 2(c) are the overall winners (this is inline with Theorem 2.2.8 as the number of holes is small). They are better strategies than simply using lexicographic order, which proves our conjecture above that enumerating type sequences in order of increasing L (the number of color changes) allows one to spot a feasible type sequence more quickly. In heavy load cases, where there are usually only few successful type sequences, the strategies 2(b) and 2(d) perform poorly. In real weather data sets, the weather constraints are typically forming large clusters, staying near to each other, and the red constraints are typically inside the blue constraints. Consequently, the red paths are more likely to be adjacent to each other so that they take more space occupied by blue constraints. Therefore, for real weather test cases, the strategies 2(b) and 2(d) perform extremely bad compared to strategies 2(a) and 2(c).

For infeasible pairs (r, b), a naive approach tests all $\binom{r+b}{r}$ type sequences. With our early termination method to do pruning of the enumeration, though, we find that, on average, a small fraction, 39.55%, of the total number of possible sequences needs to be tested before discovery of infeasibility. (It is

Data Sets	**Random Data Sets**			**Real Weather Data Sets**		
Strategy	**FCHL**	**FCLL**	**INF**	**FCHL**	**FCLL**	**INF**
(1)	9.38 (0.4s)	4.09 (0.3s)	58.48 (5.2s)	3.26 (0.3s)	3.04 (0.3s)	50.87 (3.1s)
(2a)	4.25 (0.4s)	2.27 (0.2s)	58.48 (5.2s)	3.80 (0.3s)	2.85 (0.2s)	50.87 (3.2s)
(2b)	15.28 (1.8s)	6.67 (0.5s)	58.48 (5.2s)	31.23 (2.1s)	17.13 (1.6s)	50.87 (3.2s)
(2c)	3.81 (0.3s)	2.20 (0.2s)	58.48 (5.2s)	3.15 (0.2s)	2.65 (0.2s)	50.87 (3.2s)
(2d)	15.33 (1.8s)	6.96 (0.5s)	58.48 (5.2s)	31.79 (2.1s)	17.31 (1.5s)	50.87 (3.2s)
$\binom{r+b}{b}$	245.92	55.75	320.35	957.65	212.39	4077.06

Table 2.1: The comparison among different enumeration strategies. FCHL stands for feasible cases (heavy loads), FCLL stands for feasible cases (light loads) and INF stands for infeasible test cases. We provide the average number of type sequences tested to find a feasible one or report infeasible, and the average running time of our algorithms (in seconds, in parentheses). The first column shows the strategy used, where (1) stands for lexicographically increasing order, (2a) for increasing L, the color changes, lex-increasing, (2b) for decreasing L, lex-decreasing, (2c) for increasing L, lex-decreasing, (2d) for decreasing L, lex-increasing; $\binom{r+b}{b}$ is the binomial coefficient, i.e., the overall number of possible type sequences that one would have to test using brute-force enumeration – it provides the baseline to which our methods are compared. The remaining columns show the average number of type sequences tested to find a feasible one for randomly generated data sets and real weather data sets.

easy to see that all five of the enumeration strategies search the same subset of sequences, so we do not differentiate by strategy.)

We also examined the path lengths obtained by our heuristic shortening method. We found that there is little difference among the results according to different enumeration strategies (about a 10% variability). We also found that the total length of the routes are fairly close (within about 10%-15%) to the lower bound on possible length (computed using $(r + b)$ times the length of a single shortest path from source to sink). We do not provide lengths related data because the path lengths highly depend on the size of the region of interest.

2.8 Conclusion

We considered routing multiple types of paths in a polygonal domain containing obstacles of multiple types. The very basic versions of the problem have been proved to be NP-hard. We presented approximation algorithms for different variations of the problem, as well as efficient heuristic to find the paths amidst

real-world and synthesized obstacles.

We left open approximating the maximum number of blue paths that can be routed while ensuring that a specified number of red paths exists in the domain. Another natural problem to study is maximizing the total number of all-type paths routed. For the Two Widths problem, we were not able to show hardness in the case when the width of the thinner path divides perfectly the width of the thicker ones; say, if $w_1 = 1, w_2 = 2$. On the experiments frontier, it would be interesting to investigate alternative heuristics for minimizing path lengths.

Chapter 3

Flexible Air Lane Generation to Maximize Flow Under Hard and Soft Constraints[1]

In this chapter, we consider a multicriteria optimization problem of simultaneously routing several classes of aircraft through an airspace at a fixed flight level in the presence of various types of constraints. Hard constraints are formed by hazards through which no aircraft can safely fly (e.g., severe convection, turbulence, or icing). Soft constraints are formed by hazards through which some pilots or airlines decide to fly while others do not (e.g., moderate turbulence or icing). We compute flight paths for two aircraft classes: Class-1 aircraft avoid hard constraints but are willing to fly through soft constraints, and Class-2 aircraft avoid both hard and soft constraints. Our work assists in the design of future operational concepts in which jetway routing is retired and aircraft paths are allowed to adjust to shapes and positions of constraints. We are interested in determining the capacity of an airspace with hard and soft constraints, given as input the demand profile indicating how many Class-1 and Class-2 aircraft are scheduled to enter the airspace. We report on experiments

[1]This research was funded by NASA Ames Research Center under contract NNA07BB36C for the NextGen Air Traffic Management (ATM) - Airspace Project - Subtopic 15: Translation of Weather Information to Traffic Flow Management Impact. The authors appreciate the frequent inputs from our contract monitor, Mr. William Chan from NASA Ames Research Center. Finally, we appreciate the financial support of the sponsor of the research, NASA NextGen Project Manager, Dr. Paramal Kopardekar. J. Mitchell is partially supported by the National Science Foundation (CCF-0528209, CCF-0729019). V. Polishchuk is partially supported by a personal grant from the Academy of Finland grant 118653 (ALGODAN)

both with real and with synthesized weather data.

This chapter discusses a similar set of problems as in Chapter 2, but from an empirical point of view. The chapter presents joint work with Joondong Kim, Jimmy Krozel, Joseph S. B. Mitchell, Valentin Polishchuk, and Jingyu Zou [178].

3.1 Introduction

A fundamental problem in Air Traffic Management (ATM) is estimation of the capacity of an airspace. The capacity measures the capability of the airspace to accommodate a predicted traffic demand through it, with the demand specified as the number of aircraft, of each of a set of aircraft classes, that intend to use the airspace during a given time interval. Capacity is impacted by constraints that come from various sources, including forecasted hazardous weather and planned Special Use Airspace (SUA) constraints. Such constraints may arise from typical, daily weather conditions, e.g., convective weather or turbulence, as well as from less frequent conditions such as in-flight icing, volcanic ash in the atmosphere, or other phenomena (see the survey of Krozel and Murphy [105]). The capacity is the number of aircraft of each class that can be routed through the airspace while avoiding all constraints. When the demand for an airspace exceeds its capacity, a Traffic Flow Management (TFM) strategy, such as an Airspace Flow Program (AFP) [98] [27], may be required to adjust the demand to remain at or below the estimated capacity.

We study the problem of computing flow rates for capacity estimation of an airspace at a fixed flight level in the presence of two types of constraints: *hard* and *soft*. Hard constraints are portions of airspace through which no aircraft can fly safely; these include SUA as well as severe hazardous weather phenomena (convection, turbulence, icing, or volcanic ash). Soft constraints are portions of airspace through which some aircraft classes can safely fly, while other aircraft classes may either need to avoid or choose to avoid (e.g., due to pilot or air carrier preferences). Soft constraints may arise, for instance, from certain weather hazards, such as moderate convection, turbulence, or icing. In this chapter , we concentrate on the case of *two* aircraft classes and *two* types of constraints (hard and soft); nevertheless, most of our results extend directly to multiple classes of aircraft and multiple types of constraints. (In general, the problem can be analyzed in terms of a *weather impact interaction grid*, which specifies which types of constraints must be avoided by which classes of aircraft [120].) We assume the aircraft fall into two classes: Class-1 aircraft

avoid hard constraints but are willing to fly through soft constraints, while Class-2 aircraft avoid both hard and soft constraints.

In contrast to the majority of the models used in the previous research on capacity estimation, our problem formulation recognizes the need to model weather hazards in terms of hard and soft constraints and (at least) two classes of aircraft. To the best of our knowledge, we are the first to present an algorithmic technique for capacity estimation addressing several types of constraints for multiple classes of aircraft.

Our previous work, Krozel *et al.* [101] showed that the airspace capacity depends on the ATM control laws being implemented. These laws may represent decentralized techniques, such as Free Flight [1], or centralized controls, such as flow-based routing [146] [144] [104]. In this chapter, we investigate centralized flow-based routing for two classes of flows. Our model assumes that there is a horizontal separation requirement between centerlines of flows, specified by given air traffic lane width, ω. In addition, aircraft flying along one route are separated from each other by a Miles-in-Trail (MIT) requirement. The width ω is quite general; it may depend on the class of aircraft and will, in general, be dependent on the Required Navigation Performance (RNP) of the aircraft utilizing an airlane. For current-day applications, lateral separation requirements and MIT requirements of 7, 10, 15, or 20 nmi may be appropriate; for future operations, smaller separation requirements (e.g., 5 nmi or 3 nmi) may be utilized.

The research in this chapter is in support of both the Next Generation Air Transportation System (NextGen) and the Single European Sky ATM Research (SESAR) operational concepts. The presented solution is not dependent on existing jet routes or ATM practices; our model assumes that the flight paths can be designed to pass through the airspace wherever constraints allow for feasible Class-1 or Class-2 traffic flows. Our study may help NextGen policy decision makers to determine the extent to which convection, turbulence, icing, as well as SUA may limit en route capacity in NextGen. The results establish theoretical upper bounds on capacity imposed by hard and soft constraints at a given flight level for two (or more) classes of aircraft. However, the chapter does not address the more general three-dimensional (3D) routing problem with aircraft that are climbing or descending to avoid hard or soft constraints; we leave this problem for future research.

Our model is intended to support the design of new roles for controllers and pilots in NextGen and SESAR. With this in mind, we address capacity estimation in terms of the limitations on traffic utilization based on the geometry

of the airspace and the constraints within it; we do not directly address here the limitations on traffic utilization imposed by workload considerations of the people that monitor the airspace. In particular, we are not addressing maximum aircraft count per sector (e.g., Monitor Alert Parameter (MAP) values).

3.1.1 Related Work

In [104], we reported on capacity estimation techniques for airspaces with convective weather constraints (of a single type); related prior research surveyed in that paper includes [152] [157] [47] and [34]. An experimental comparison of techniques for estimating the sector capacity given convective weather constraints in today's ATM system is presented in [159].

In NextGen we expect that jet routes can be dynamically redefined to adjust flows of traffic around weather constraints, and that controller workload will not be a limiting factor. In such a setting, the maximum capacity of an airspace can be assessed using extensions of maximum flow theory in networks (see, e.g., [6]) to maximum flows in geometric domains. The standard network MaxFlow/MinCut Theorem extends to geometric domains [161]; its algorithmic properties have been studied originally in [127] and more recently in [133] and [14]. The MaxFlow/MinCut theory has been applied in ATM capacity estimation tasks, for determining the maximum throughput across an en route airspace with hard constraints given either by a traffic flow pattern [159], or by a uniform distribution of flow monotonically traversing in a standard direction (e.g., East-to-West), or random, Free Flight conditions [104]. The maximum capacity of terminal airspace may also be determined by transforming the problem into an effectively two-dimensional (2D) domain on the ascent or descent cone modeling the transition airspace [99].

We know of no prior algorithmic results directly related to the multi-class geometric flow routing problem studied in this chapter. At the same time, there is an abundance of work on computing multiple paths and flows (of a single class) in geometric domains. In [170], a classification of existing approaches to multiple paths planning is suggested. In particular, in the prioritized planning, the paths are found iteratively, one by one, treating already routed paths as obstacles for each newly added path. This is the same strategy as employed in [40] [146] and [144], who developed routing algorithms for multiple aircraft based on iterative packing in space-time. Apart from the prioritized routing, other routing schemes include centralized path planning, roadmap-based methods, and decoupled path planning [170]. In [14],

a pseudopolynomial-time dual-approximation algorithm was given to determine a maximum number of trajectories for velocity-bounded agents (disks) moving in a polygonal domain in which there are moving (polygonal) constraints. In ATM terms, the diameter of the disks corresponds to lane width or the horizontal separation standard.

3.1.2 Chapter Organization

We first discuss the modeling of the problem. We then review the theory of capacity estimation. Next, we present our main contribution - algorithmic methods for capacity estimation in presence of hard and soft constraints, and applications of our methods to icing and turbulence constraints. We conclude with a discussion of our results and future work.

3.2 Modeling

To motivate our model, we present two examples of weather impacts on the National Airspace System (NAS) - turbulence and icing.

The first example is turbulence. Krozel *et al.* [109] studied the rules and regulations associated with turbulence, and have found that there are two significant levels of turbulence that determine hard and soft constraints. Moderate-or-Greater (MoG) turbulence tends to limit the capacity of en route airspace since passenger comfort and safety is a high priority for many airlines; thus, many airlines choose to avoid MoG turbulence. Still, some aircraft and airlines do fly through MoG turbulence; for instance, ferry flights, cargo flights, and some business jets. However, if Severe-or-Greater (SoG) turbulence is forecast or reported, it poses an immediate safety hazard, which closes airspace and, if encountered, may require diversion due to injuries and/or required aircraft inspections. SoG turbulence can cause aircraft structural damage/failure, loss of control, and injury or death to passengers. Thus, SoG turbulence areas of the NAS are not safe for flight, and therefore represent hard constraints.

The second example is in-flight icing. Krishna and Krozel [97] studied the rules and regulations as well as pilot/aircraft/airline responses (e.g., cancellations, en route holding, altitude deviations, and diversions) related to significant in-flight icing events that result in SIGMETs (Significant Meteorological Information). SIGMET airspace regions severely restrict the flow of traffic through a region of airspace described by the horizontal polygon

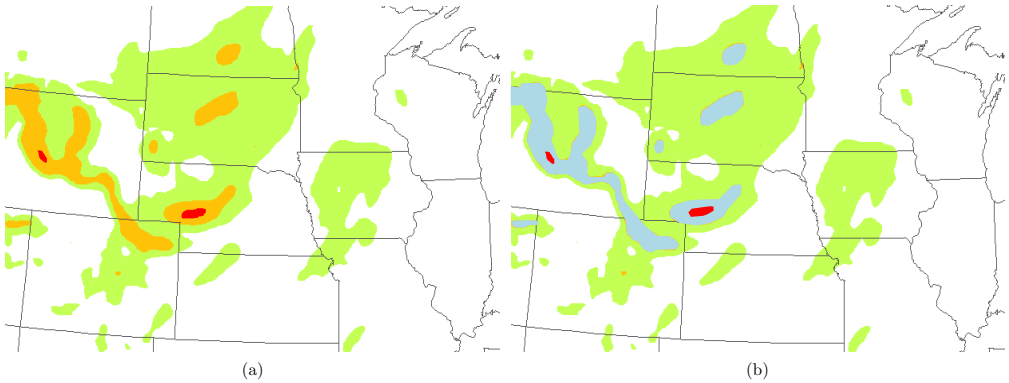


Figure 3.1: Mapping forecasts into hard and soft constraints. (a) GTG flight level turbulence data. Red: areas with severe turbulence; Orange: areas with moderate turbulence; Green: areas with light turbulence. (b) Hard and soft constraints model. We model the red areas as hard constraints (still shown in red) and the orange areas as soft constraints (shown in light blue).

boundaries and the lower and upper altitude limits. The SIGMET generally describes a hard constraint region due to the severity of the icing potential within it, which is, generally, a SoG icing level. For MoG icing levels, aircraft and airlines may enter the airspace; however, the decision generally depends on the aircraft and equipment. If aircraft are equipped to address icing conditions, then they may flight through MoG icing. However, other aircraft, e.g., General Aviation (GA) aircraft, may not be equipped to fly through icing, in which case even MoG icing regions should be avoided. In-flight icing is thus determined by MoG or SoG icing severity levels and the pilot, airline policy or aircraft capability.

Fig.3.1 illustrates the modeling of turbulence, showing the conversion of a turbulence forecast (Graphical Turbulence Guidance (GTG)) into hard and soft constraints.

3.2.1 Airspace Model

The traffic flow at a fixed flight level is modeled as a 2D problem. The airspace is modeled by a polygonal domain P, representing a NAS sector, center, or Flow Evaluation Area (FEA). The traffic enters P through a specified *source* edge, and exists through a *sink* edge. Envisioning West-to-East traffic, assume that the source and the sink are the "left" and the "right" edges of P (Fig.3.2). Furthermore, we do not consider flows of aircraft originating or terminating

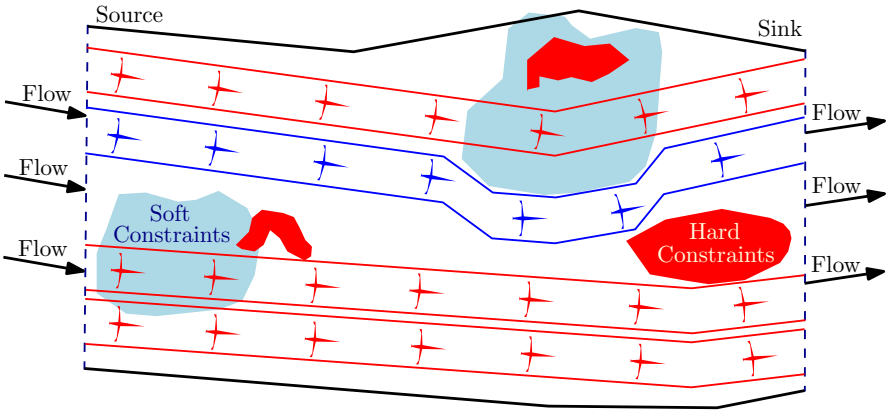


Figure 3.2: Airlane packing for two classes of aircraft among hard and soft constraints. Class-1 (red) air lanes avoid only hard constraints while class-2 (blue) air lanes avoid both hard and soft constraints. In the example presented, while one could route 5 class-1 air lanes, once a class-2 air lane is established, there is capacity for only 3 class-1 air lanes. (The class sequence from bottom to top is class 1, class 1, class 2, class 1.)

within airspace P.

For throughput calculations, aircraft within the airspace are assumed to have a constant speed along each flow. While our model allows speeds to be different on different flows, our experiments assumed all aircraft had the same speed (we used 420 nmi/h). Additionally, the model allows for any specified horizontal separation requirement (which determines the airlane width, ω); for our experiments, we assumed a horizontal separation requirement of 5 nmi for en route airspace. One may also specify an additional safety margin, δ with respect to hazardous weather constraints; Airlanes are required to be at least distance δ from a constraint that it must avoid (we used δ =0). For simplicity, we do not account for the earth's curvature.

3.2.2 Hard and Soft Constraints

The severity of hazardous weather may be quantified by an intensity threshold in the National Weather Service (NWS) scale and a clearance level over the echo top height - the Convective Weather Avoidance Model (CWAM) [157] [47] [34] and [159]. Regions of high convective weather intensity, such as CWAM Weather Avoidance Fields (WAFs), define airspace constraints. However, Kuhn [112] suggests that there is no single threshold that defines a constraint region for

all aircraft; instead, there is a need for more than one threshold to reflect different aircraft/pilot behavior with respect to convective weather constraints. This gives rise to the modeling of hard and soft constraint thresholds for TFM planning in the presence of convective weather: Regions with intensity above a threshold, τ_{hard},define hard constraints, while those regions with intensity above a threshold, τ_{soft}, but below τ_{hard} ($\tau_{soft} < \tau_{hard}$), define soft constraints. Similarly, there are different levels of severity for turbulence and for icing, and these, together with pilot/airline preferences and aircraft type, determine regions defining hard or soft constraints. For example, as illustrated in Fig.3.1, the classification of turbulence or icing as SoG may define hard constraints, and the classification as MoG may define soft constraints.

Our analysis is with respect to short time intervals over which the forecast accuracy justifies the assumption that weather cells form static constraint regions. Yet, we discuss issues related to forecast uncertainty, and apply our methods to yield a stochastic throughput analysis. We leave for future work the extension of our methods to dynamic forecasts, for which we want to account explicitly for the time-varying nature of the weather constraints.

3.2.3 Objective

Our goal is to determine maximum throughput for a demand that consists of a mixture of Class-1 and Class-2 aircraft that are to cross airspace P from source to sink. Class-1 aircraft avoid hard constraints but are willing to fly through soft constraints, while Class-2 aircraft avoid both hard and soft constraints. Our formal goal is to establish if it is possible to route a specified number, I, of Class-1 airlanes and a specified number, J, of Class-2 airlanes. Each airlane is a thick path, whose centerline represents a route and whose width w represents the accuracy with which aircraft are expected to be able to navigate the route. Airlanes must be pairwise-disjoint and not overlapping the set of constraints relevant to the traffic utilizing the airlane (hard constraints for Class-1, and hard and soft constraints for Class-2). Thus, our problem is that of packing within the airspace a set of airlanes of two classes: I airlanes of Class 1 and J airlanes of Class 2, with each set of airlanes satisfying its corresponding set of constraints. Maximizing the throughput is a bicriteria optimization problem: one may want to maximize the number of Class-1 airlanes subject to a lower bound on the number of Class-2 airlanes or to maximize the number of Class-2 airlanes subject to a lower bound on the number of Class-1 airlanes. We provide algorithms that serve as centralized strategies for this optimization problem.

3.3 Maximum Flow Rate Theory

We now review the theoretical solutions to the problem of computing a maximum flow rate from source to sink through an airspace.

3.3.1 Flows in Discrete Networks

Recall some basic notions and results about flows in graphs. A *directed graph*, G=(N,A), consists of a set N of *nodes* and a set A of (directed) *arcs* (or *edges*) connecting pairs of nodes. A *flow network* is a directed graph G in which each edge e has a *capacity*, *c(e)*, and two nodes of N are designated as the *source* and the *sink*. All other nodes of N are *internal nodes*. A *flow* in G is an assignment of a *flow value*, $f(e) \leq c(e)$, to each edge e in A, such that for each internal node v the flow through v is *conserved*: the sum of the flows on edges going into v is equal to the sum of the flows on edges going out of v. The *value of the flow* in G is defined to be the total flow out of the source node, which equals (by flow conservation at internal nodes) the total flow into the sink node. In the *maxflow* problem, one seeks a flow with maximum value. A *cut* is a partition of the nodes N into two sets, X and Y, such that the source is in X and the sink is in Y. An edge e=*(u,v)* crosses the cut if $u \in X$ and $v \in Y$. The capacity of the cut is the sum of the capacities of the crossing edges. The capacity measures the maximum amount of net flow possible from the source to the sink. A fundamental result in network flows is that *maxflow* equals *mincut*: the maximum flow possible is equal to the capacity of a minimum-capacity cut. This "*maxflow/mincut*" theorem is a manifestation of "duality" in linear programming and optimization theory; see, e.g., [6]. Efficient (polynomial-time) algorithms are known for computing maximum flows and minimum cuts in networks.

The notion of flow in a discrete network can be extended to a continuous geometric domain P [161] [127]. Similar to the source and sink nodes in discrete networks, two edges on the boundary of the outer polygon of P are designated as the source and sink. Now, the source and sink edges split the outer boundary into two parts: top T and bottom B. More precisely, the outer boundary of P is partitioned into four portions, appearing in the clockwise order: bottom, source, top, and sink. Polygonal domain P contains constraints; the constraints are pairwise-disjoint simple polygons (Fig.3.3).

As with networks, a *cut* in P is a partition of the domain into two parts such that the source edge is in one part, and the sink edge is in the other; the *capacity* of the cut is the length of the common boundary of the parts. With a

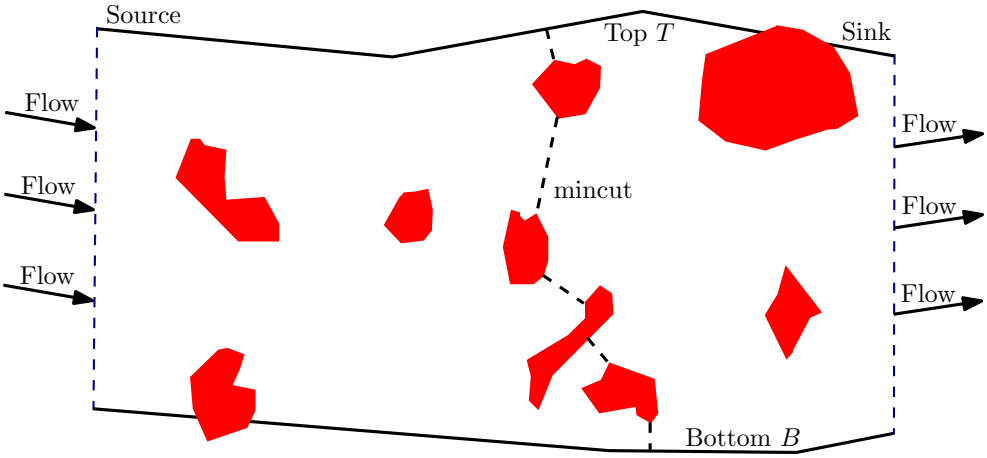


Figure 3.3: The Min-Cut defined by the source, sink, hazards, and sector geometry.

slight abuse of terminology, we usually refer to this common boundary as the cut. A *mincut* is a cut of minimum length. A mincut is therefore a sequence of line segments, connecting B to T, "hopping" from constraint to constraint in such a way as to minimize the total distance travelled within P(Fig. 3.3). Strang [161] shows that, as in discrete networks, the maxflow/mincut theorem also holds for geometric domains, where flow refers to a divergence-free vector field in P, and the flow value is defined as the path integral, along the source, of the inner product of the flow field with an inward-pointing unit normal vector. (The divergence-free property of the flow field is the analogue of the flow conservation at internal nodes of a network.) Mitchell [127] develops algorithms to compute a mincut efficiently in 2D polygonal domains using geometric shortest path techniques (the "continuous Dijkstra" paradigm); his algorithm also produces a flow field, which can be envisioned as a continuum of *flowlines* from source to sink. The mincut is the "bottleneck" to fluid flow from source to sink; its length quantifies the maximum achievable flow rate.

In order to apply the theory of continuous flows to our ATM model, we consider a variant of the theory, in which flowlines are grouped into discrete bundles of *thick paths* (airlanes of width w). The maxflow/mincut theory is extended to compute the maximum number of *thick paths* across the domain, from source to sink [14]. Formally, a *thick path* is the Minkowski sum of a (thin) path, from a point on the source to a point on the sink, and a disk of diameter ω centered at the origin. (E.g., if the thin path is polygonal, the thick path is the union of a set of rectangles of width ω - one per edge of the centerline path,

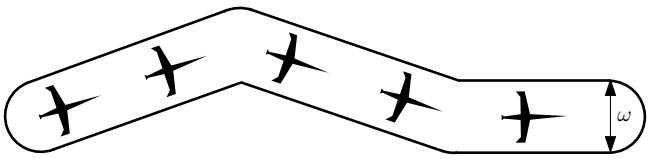


Figure 3.4: A thick path defines an airlane of width ω.

and a set of circles of diameter ω - one per turn point of the centerline path). See Fig.3.4. The discrete maxflow problem in P is to compute a maximum number of pairwise-disjoint thick paths within P from source to sink. The maxflow/mincut theory extends to this "discrete" variant of the continuous maxflow problem [14].

3.3.2 Multi-Class Throughput Problem Formulation

We now describe a new problem, which generalizes the discrete maxflow in geometric domains to multi-class aircraft routing.

As before, the input to our problem is a polygonal domain P with a source and a sink edges on the boundary, and a set of polygonal constraints. The new aspect of the problem is that the constraints are of K types and the sought thick paths are of M classes. The width of a class-m path is ω_m, m= *1,2,3,...,M.* A path of class m must avoid the constraints of types $O_m \subseteq \{1,...,K\}$, but can pass through the constraints of the other types. (The specification of the sets O_m for each m gives the *weather impact interaction grid*, as discussed in [120].) The capacity estimation problem is: Given integers n_1,n_2,n_3,...,n_M,decide if there exist n_1 class-1 paths, n_2 class-2 paths,..., and n_M class-M paths from source to sink through P, with no overlap among the (thick) paths, and each class-m path avoiding constraints of types in the set O_m.

While the model just described applies to a wide range of possible types of constraints and aircraft classes and interactions between them, our focus for the remainder of the chapter is on the case of just two types of constraints (i.e., K=2) and two classes of aircraft (i.e., M=2). Specifically, in our airlane routing problem, the constraints are either *hard* (type 1) or *soft* (type 2). The two classes of aircraft/paths, Class-1 and Class-2, have the sets of obstacles specified by $O_1 = 1$ and $O_2 = 1, 2$, indicating that Class-1 aircraft avoid hard constraints (but can travel through soft constraints), while Class-2 aircraft avoid both hard and soft constraints.

3.3.3 Class Sequences

For any set of airlanes through P linking the source to the sink, the airlanes are ordered from bottom B to top T, with the ith airlane being the one with *(i-1)* airlanes below it. The specification of the ordered list of path classes is a class sequence, C=(c_1,c_2, ...), where each c_i is one of the M classes of airlanes. Here, M=2, so each class sequence is a sequence of 1's and 2's, which we often denote without the parentheses and commas (e.g., 11222122 instead of (1,1,2,2,2,1,2,2). For example, Fig.3.2 illustrates the class sequence C=1121.

3.3.4 Bottommost Paths

Paths (airlanes) p_1, p_2, ..., pm are called *bottommost* paths if p_1 runs "as close as possible" (in a sense made rigorous in [14]), to the bottom B, and p_i runs as close as possible to p_{i-1}, for i=2,3,..,m.

3.3.5 Theoretical Results

Kim *et al.* [90] prove that if there exist m paths with class sequence C, then there exist m bottommost paths with the same class sequence C, and the paths can be found efficiently by a "bottommost filling of the domain" with the paths. Therefore, if the class sequence C is given, the capacity estimation problem can be solved efficiently by computing bottommost paths. However, Kim *et al.* [90] also show that the problem is NP-hard if the type sequence is not known: it is NP-hard to decide if it is possible to route I lanes of Class 1 and J lanes of Class 2. An alternation in a class sequence C is a place where C changes from 1 to 2 or vice versa (note that the number of alternations, n_a, can be larger than the number of classes, e.g., the sequence 1211221212 has $n_a = 7$); For cases in which when the maximum number, n_a , of alternations in a class sequence is small (e.g., 111221111 has only $n_a = 2$ alternations, when there is a change from 1 to 2 or vice versa), the problem can be solved exactly in polynomial time, where with the exponent in the running time dependings on n_a. Further, from the point of view of approximation algorithms, Kim *et al.* [90] shows that for the problem in which there are two classes of paths and two types of constraints, one can obtain a provable approximation to the optimal number of Class-1 and Class-2 paths.

3.4 Algorithmic Solution Approaches

Next, we outline a routing algorithm that solves the problem: Given two integers, I and J, route I Class-1 paths and J thick Class-2 paths, each having width ω, in a polygonal region P populated with hard and soft constraints. For problems with more than two classes/types of thick paths and constraints, with possibly different thicknesses, our methods extend to yield a similar solution.

Our algorithm works by scrolling though class sequences. For each class sequence C we use an efficient way to either compute the paths with the class sequence given by C (in which case the algorithm terminates with a success), or to determine that no collection of paths with sequence C exists (in which case we proceed to the next sequence). The total number of different sequences equals to $(I+J)!/(I!J!)$ which is the number of ways to choose I places for 1 and J places for 2 in a sequence of 1's and 2's of total length $I+J$; here $N!$ denotes N-factorial - the product of all integers from 1 to N. This number grows fast as a function of I and J; hence, we use several heuristics (described below) to guide the scrolling and prune large fraction of infeasible sequences.

For any particular given class sequence C, we find the paths by a bottommost fill, the details for which are presented next.

3.4.1 Bottommost Path Filling Algorithm

Rather than working with the continuum of all possible thick paths, we take a practical approach to generating bottommost paths that are restricted to a discrete set of possibilities, by searching for paths within a regular square grid V inside P. We add to V a group of source and sink nodes, which are discrete points uniformly distributed along the source and sink edges. See Fig.3.5.

A directed graph $G=(V,E)$ is formed. The edge set E contains directed edges (p,q) connecting points $p=(p_x, p_y)$ and $q=(q_x, q_y)$ in V whenever q is to the right of p, and is "close" to p; specifically, we require that $1 \leq q_x - p_x \leq D$, using $D=5$ (a default parameter). We keep only those segments pq as edges in E, for which no point of pq lies within distance $\omega/2$ of any hard constraint. Moreover, within the set E, we mark an edge pq as a Class-2 edge if, in addition, no point of pq lies within distance $\omega/2$ of any soft constraint. This models the fact that all edges are feasible for Class-1 aircraft, while only Class-2 edges are feasible for Class-2 aircraft.

In our bottommost path filling algorithm, we compute bottommost paths from source nodes to sink nodes within graph G, according to the path class. Since our directed edges are all oriented from left to right (recall that $1 \leq q_x - p_x$

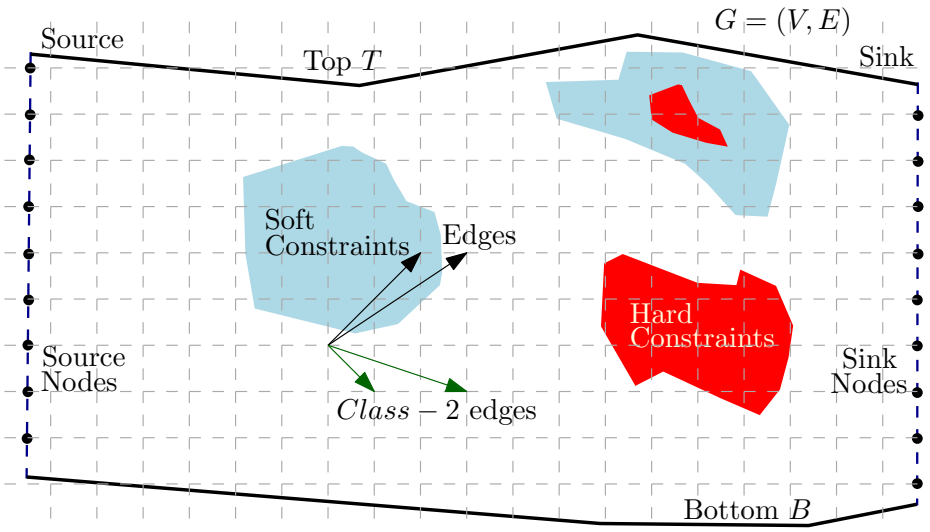


Figure 3.5: Graph G used in the bottommost path filling algorithm. Vertex set V is composed of grid points (intersections of dashed grey lines) and nodes along the source and sink edges.

for edge (p,q)), the computed paths are necessarily x-monotone; this enforces that the paths make progress in the direction from source to sink (left to right), without doubling back. One bottommost path is routed by a Depth-First Search (DFS) [43] within G. Specifically, the path originates at the bottommost feasible source node, and progresses monotonically to the right, giving preference to the bottommost (most clockwise) feasible edges leading out of a node. The DFS succeeds in finding a path if it reaches a sink node; otherwise, the search from the source node fails, and we move to routing from the next available source node. Fig. 3.6 shows an execution of the algorithm for the sequence "121". At the highlighted nodes in the search, there is branching, as the depth-first search has to retrace its steps and choose a different branch, after having hitting a "dead end".

At the completion of the bottommost fill, we either have a sequence of I Class-1 and J Class-2 paths as desired, or we know that the current choice of class sequence is infeasible, so the algorithm moves on to the next class sequence.

Sequence Pruning. Our algorithm maintains a black list of infeasible subsequences. Sequences that begin with a *black-listed* subsequence are immediately ruled out as infeasible. For instance, if sequence "1212212121" fails after testing only "1212", then "1212" is added to the black list so that all

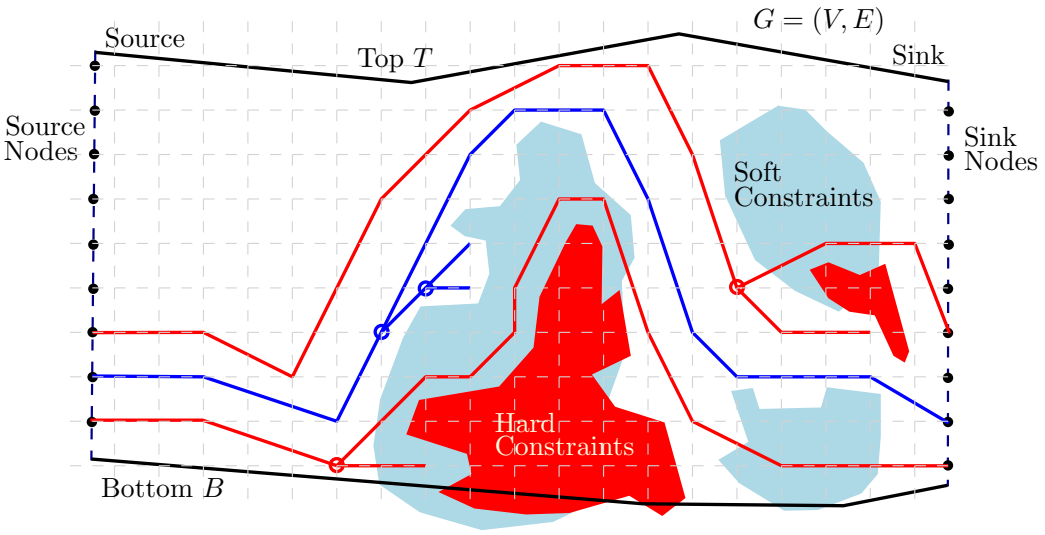


Figure 3.6: The bottommost fill algorithm for sequence "121". Small circles represent branch backtrack points in the depth-first search.

permutations starting with "1212" are immediately recognized as infeasible (without having to perform a bottommost fill search).

Leftover Space Testing. After routing a bottommost path, our algorithm computes two mincuts in the leftover space P_l above the routed paths in P: the mincut m_1 in P_l with only hard constraints and the mincut m_2 in P_l with both hard and soft constraints. We use the capacities m_1 and m_2 to judge if the leftover space can accommodate the remaining Class-1 paths and Class-2 paths. For instance, if we have successfully routed already i¡I Class-1 paths and j¡J Class-2 paths, then we know that the remaining $(I\text{-}i)+(J\text{-}j)$ paths (of either class) cannot be routed if m_1¡ $(I\text{-}i)+(J\text{-}j)$; this is because m_1 is the maximum number of thick paths that can be routed through P_l. Similarly, we know that the remaining $(J\text{-}j)$ Class-2 paths yet to be routed cannot exist in P_l if m_2¡$J\text{-}j$; this is because m_2 is the maximum number of thick paths that can be routed through the remaining airspace while avoiding both hard and soft constraints.

Strategic Enumeration of Class Sequences. The order in which class sequences are explored makes a considerable difference in the speed with which our algorithm finds a routable class sequence, if one exists. (Recall that the algorithm concludes once a first feasible class sequence is discovered, for which a bottommost fill succeeds.) Instead of enumerating class sequences in arbitrary order or in a natural lexicographic order, our algorithm uses the order of

increasing number of class alternations. For example, if we are searching for I=5 Class-1 paths and J=5 Class-2 paths, then lexicographic ordering would explore (1111122222, 1111212222, ...), while enumeration by increasing number of class alternations would explore (1111122222, 2222211111, ...). Kim *et al.* [90] experimentally proved that this enumeration strategy helps find a feasible class sequence much faster than some competing alternatives.

3.4.2 Postprocessing: Tautening Bottommost Paths

The bottommost paths are generally unreasonably long and are unrealistic for ATM. Hence after the bottommost paths are routed, we "pull them taut" using an iterative local shortening heuristic. Specifically, take the last-but-one topmost path p_{I+J-1}, and declare it as an obstacle. Then replace the topmost path p_{I+J} with the shortest path $p*_{I+J}$ routed in the space between the top T and p_{I+J-1} (of course, we make sure $p*_{I+J}$ is a feasible path by not letting it penetrate any constraints). Next, declare $p*_{I+J}$ and p_{I+J-2} as obstacles, and replace p_{I+J-1} with the shortest path p_{I+J-1} between $p*_{I+J}$ and p_{I+J-2}. Continuing this way, we obtain a set of $I+J$ short paths $p*_{I+J}$,...,$p*_l$. Our experimental results indicate that this taughtening heuristic is very useful in producing flyable, monotonically increasing paths in the direction from source to sink (see Fig. 3.7(d,f) and Fig. 3.8(c)).

3.5 Experiments

We applied our solutions from the previous section to two sets of data. The first is the real-world weather forecast maps based on GTG and CIP forecasts over the NAS at 3:00 pm on Jan. 24, 2007 at 38000 ft. The second set is based on synthesized weather data that simulates real weather data and allows us to investigate the effectiveness of our algorithms on a broader class of constraints than is readily available in selected real datasets.

3.5.1 Real Weather Data

In our experiments with real weather data, the FEA is a rectangular domain extracted from a GTG map; the FEA covers a portion of Midwest of the US. Aircraft fly through the FEA from West to East; i.e., the source is the west side of the rectangle and the sink is the right side. (For our analysis it would not matter if the roles of source and sink are reversed.) The distance from

the northern to the southern boundary of the FEA is 313 nmi. We assume that both Class-1 and Class-2 airlanes have width 8 nmi, with an additional separation of 8 nmi between airlanes; thus, the total distance from centerline to centerline of two adjacent airlanes is 16 nmi.

We conducted three experiments: (1) Given two integers I and J, as well as a designated class sequence C of I 1's and J 2's, determine if I Class-1 and J Class-2 airlanes can be routed from source to sink according to C. (2) Given integers I and J, but no class sequence, determine if I Class-1 and J Class-2 airlanes can be routed from source to sink. (3) Given prescribed entry and exit points on the source and sink edges, as well as integers I and J, determine if there exist I Class-1 and J Class-2 airlanes between the entry and the exit points. Results of these experiments are shown in Fig.3.7.

3.5.2 Experimental Results

We tested over 2000 test cases, including both real weather data and synthesized weather data cases. Results show that our capacity estimation algorithm is practical and efficient. The algorithm's implementation is fast because it tests far fewer than the maximum possible number of sequences. For almost all test cases where the given lanes are routable, the algorithm tests only very few permutations (less than 10), even for very large scenarios, e.g. $I + J > 100$. The test for each permutation is very fast: For a rectangular region partitioned into a base grid of 125-by- 63 cells, in which there are 5 obstacles of average size (area) 176 cells, it takes 11.61ms, on average, to test a specific class sequence (using a 2.6G Intel CPU). For smaller regions that have sparse obstacles, e.g. with a base grid of 40-by-40 cells (a resolution that likely suffices for an airspace sector), it takes on average 3.32ms to test one class sequence. Therefore, for routable test cases, the algorithm usually reports results very quickly (less than 4 seconds even for very large instances).

For fail-to-route test cases, if there are only sparse obstacles, the algorithm usually returns "unroutable"very quickly (within 5 seconds), thanks to leftover space testing and sequence pruning. But there are extremely slow fail-to-route cases for which the algorithm must test up to 20% of the maximum number, $(I+J)!/(I!J!)$, of class sequences. (This, of course, is expected, since we know the problem is NP-hard.) Fortunately, for cases in which $I + J < 20$, the algorithm was always able to report the result in less than 120 seconds, in all of the experiments.

The path tautening phase of the algorithm was found, on average, to take, per path being optimized, 2.16 s for the smaller instances (based on a 40-by-40

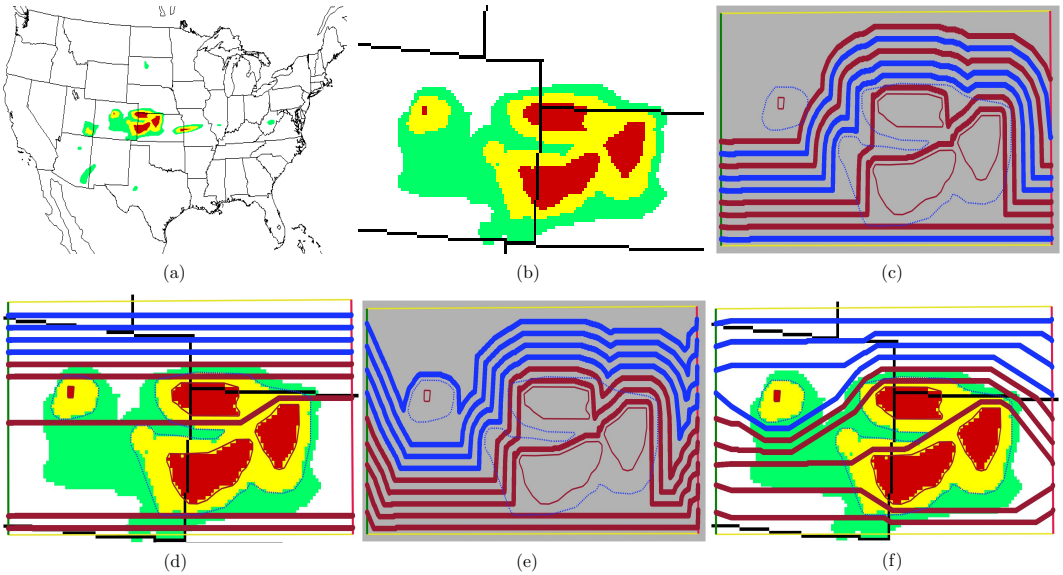


Figure 3.7: Experiments using real weather, with I=5, J=4. (a) The GTG map. (b) FEA and the constraints. (c) An example of bottommost paths. (d) Pulled taut paths. (e) Bottommost paths with given entry and exit nodes. (f) Pulling taut the paths shown in (e).

grid of cells), and 7.97 s for the large instances (based on a 125-by-63 grid of cells). Examples of paths after tautening are shown in Fig. 3.7(d,f) and Fig. 3.9(c); the resulting paths appear to be reasonable for ATM applications.

3.5.3 Probabilistic Weather Maps

So far we assumed that the position and size of the constraints in the airspace were known precisely in advance, with no source of randomness. While this assumption may be valid over short time horizons, when forecast data is highly accurate, it is more realistic to expect that weather data is described by a *stochastic* forecast, perhaps represented as a weighted ensemble set of forecasts (with weights corresponding to probabilities or "beliefs"). The ensembles may, e.g., be generated by numerical prediction models with randomly perturbed initial conditions, by multiple models applied to the same initial conditions, or by a data assimilation process. In [134], maximum throughput and capacity estimation is studied for stochastic weather models based on an ensemble of forecasts. Specifically, for each forecast, the mincut is computed and the probability distribution of the mincut value is calculated explicitly, based on

the probabilities associated with members of the ensemble. More sophisticated stochastic weather models, e.g., based on probabilistic weather maps and ensembles of probabilistic weather maps, can also be considered, together with an explicit modeling of the spatial correlation between nearby points.

These methods can form the basis of a TFM strategy for mixed equipage aircraft classes involved in an AFP [98]. Specifically, for a given probabilistic weather map associated with a look-ahead time (e.g., 1-h, 2-h, or 3-h forecasts), predicting levels of turbulence and/or icing, we can compute the probability of being able to route I airlanes for Class-1 traffic and J airlanes for Class-2 traffic. Then, based on how high these probabilities are, a threshold-based policy can be used to decide whether to continue flights on their predicted flow across a FEA, or to reroute the flights around the FEA, or to perform an altitude change (based on computing the probabilities of routability at other altitudes using our capacity algorithm). That is, we convert a probabilistic forecast to an ensemble of forecasts, each with an associated probability. Then, we run our deterministic algorithm for each member of the ensemble, and compile the data to determine the probability of successful routing over a full set of ensemble forecasts.

In our experiments, given one forecast for the region of interest P, which yields a set of hard/soft constraints (based on MoG or SoG levels of turbulence or icing), we generate an ensemble of forecasts to represent a stochastic forecast model. The generation is based on randomly selecting a seed point q inside P. If q is within a constraint (hard or soft), we place, with probability p (a user-defined parameter, close to 1) a random polygonal constraint centered at q. (In our experiments, we used random quadrilaterals, generated by selecting four points at random in the four quadrants centered at q; other random shapes are possible too, e.g., random disks, polygons, etc.) If the seed point q falls outside all constraints then random polygonal constraint is placed at q with probability *1-p*. This way we obtain a set of forecasts each looking similar to the nominal input forecast.

The test results of the probabilistic weather maps are shown in Fig. 3.8. Based on the same weather map that was used in Fig.3.7, we generate 1000 random instances of a weather forecast and test the probability that I Class-1 and J Class-2 airlanes can be routed through the FEA by executing the algorithm on each generated instance (member of a synthetic ensemble).

Additional results for capacity estimation based on synthesized weather data are shown in Fig.3.9. The algorithm succeeds upon discovering the routable class sequence "111122222211112", which is found quickly, since it has only 3

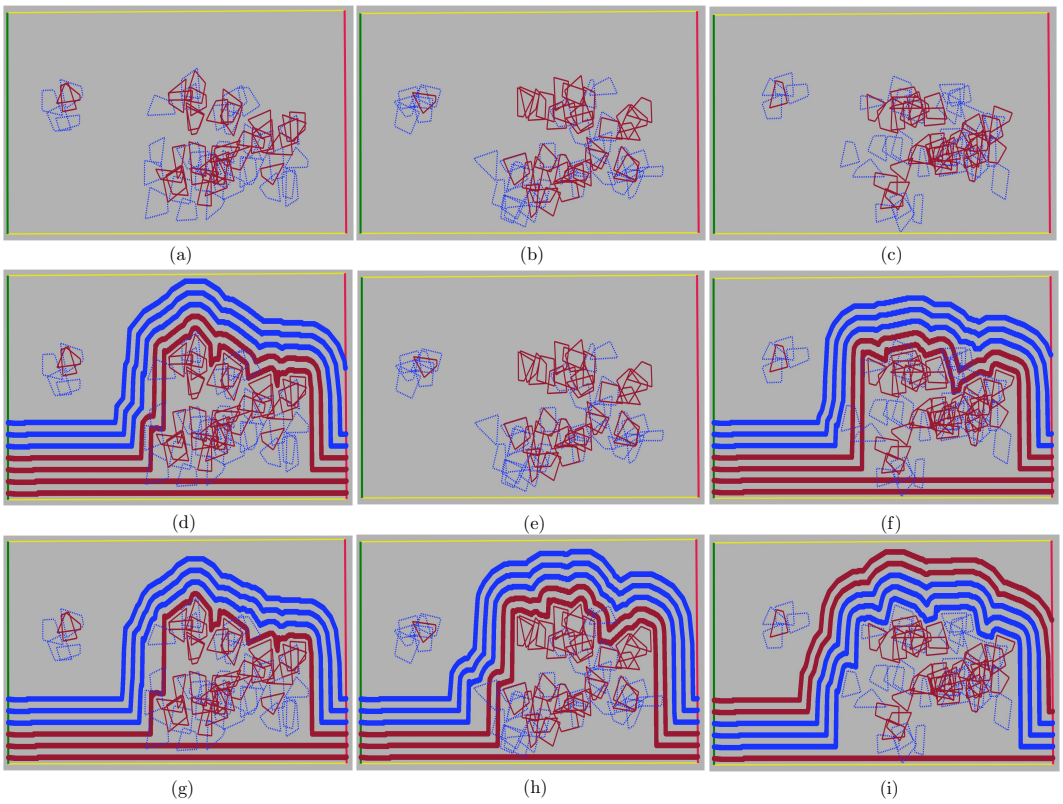


Figure 3.8: Paths computed using probabilistic weather maps. (a), (b) and (c) present 3 randomly generated weather maps. (d), (e) and (f) shows the bottommost paths routed in (a), (b), (c) respectively, when the number of class-1 lanes, I, is 5 and that of class-2 lanes, J, is 3. Bottommost routing in instance shown in (b) failed. (g), (h) and (i) shows the routed bottommost paths when $I = J = 3$.

alternations.

3.6 Conclusion

We present a mathematical model and an algorithmic method for capacity estimation in airspaces with hard and soft constraints for two classes of aircraft. We discuss related theoretical results and propose a practical algorithm for throughput computation and routing of two-class airlanes among hard and soft constraints. Further, we demonstrate results from these algorithms and

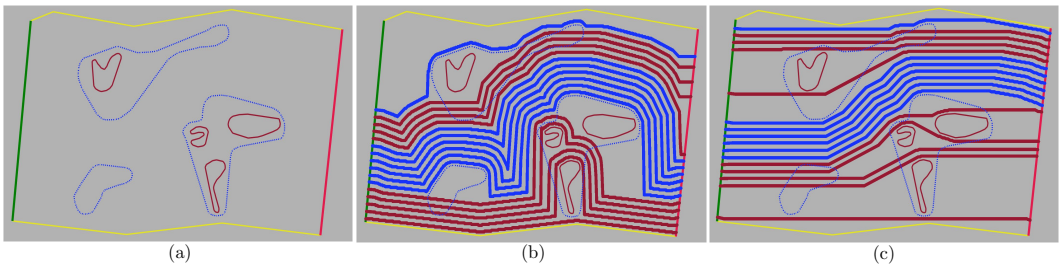


Figure 3.9: Synthesized weather data, with I=9, J=7. (a) The airspace model. (b) Bottommost paths. (c) Pulled taut paths.

report on experience in applying them to perform capacity estimation using deterministic and probabilistic weather maps based on real weather data as well as on synthetic data. The algorithms presented here are well grounded in theory and are shown experimentally to be practical and efficient.

3.7 Future Research

Future research includes:

1. Better approximation algorithms for capacity estimation in the presence of hard and soft constraints. Can we obtain an approximation algorithm for maximizing the number, I, of Class-1 airlanes, given a lower bound on the number, J, of Class-2 airlanes? (Known approximation algorithms [90] relax the optimality of the number of airlanes of both classes.)

2. Allowing non-monotone airlanes. (Our implementation is based on an algorithm that searches over sets of x-monotone airlanes.)

3. 3D airspaces, allowing multiple altitudes and climb/descent profiles.

4. Multiple sources/sinks on the boundary of the airspace and to crossing patterns of traffic, in which the demand includes flows of aircraft that must cross. This introduces scheduling into the problem, and the search becomes one of computing a maximum number of thickened "tubes" in space-time, with tubes of different classes, corresponding to the aircraft capabilities with respect to hard and soft constraints. Our current techniques do not extend straightforwardly to this scenario.

5. Three or more types of constraints ($K3$) and three or more aircraft classes ($M3$). This will allow us to compute capacities for multiple classes of aircraft whose constraints are determined by a general weather impact interaction grid [120].

6. Modeling limitations that come from controller workload for monitoring the airspace.

7. Dynamic aspects of the problem, including: moving weather cells, changing traffic composition, and organizing classes of traffic prior to entering into a FCA. Our methods assume that the traffic mix is known (or at least estimated) over a given time horizon. In order to address general changes in traffic mix over a planning time horizon, it is necessary to solve the problem in the space-time domain, e.g., using techniques of [14]; these techniques have not yet been generalized to the mixed equipage domain, though, so this remains a topic for future research. If weather cells move, particularly if they move "as a whole", then it would be advantageous to extend our work to design "flexible" airlanes, as has been studied recently by Krozel *et al.* [178] in the case of a single class of aircraft.

8. Introduction of a cost function. While our post-processing method of pulling paths taut is a heuristic intended to produce short paths, it does not come with theoretical guarantees on the optimality of the set of paths. The problem of computing an "optimal" set of (thick) paths in a domain is known as the geometric minimum cost flow problem, which has been studied by Mitchell and Polishuchuk [133] for single class flows. Future work will examine the extension of that theory to hard/soft constraints and multiclass (multicommodity) flows.

Chapter 4

Convex Transversals[1]

In this chapter, we address the question initially posed by Arik Tamir at the Fourth NYU Computational Geometry Day (March, 1987): "Given a collection of compact sets, can one decide in polynomial time whether there exists a convex body whose boundary intersects every set in the collection?"

We prove that when the sets are segments in the plane, deciding existence of the convex stabber is NP-hard. The problem remains NP-hard if the sets are scaled copies of a convex polygon. We also show that in 3D the stabbing problem is hard when the sets are balls. On the positive side, we give a polynomial-time algorithm to find a convex transversal of a maximum number of pairwise-disjoint segments (or convex polygons) in 2D if the vertices of the transversal are restricted to a given set of points.

We also consider stabbing with vertices of a regular polygon – a problem closely related to approximate symmetry detection.

This chapter presents joint work with Esther M. Arkin, Claudia Dieckmanny, Christian Knauerz, Joseph S. B. Mitchell, Valentin Polishchuk and Lena Schlipf [13].

[1]We thank the reviewers for helpful comments. E. Arkin, J. Mitchell, and S. Yang are partially supported by the National Science Foundation (CCF-0729019, CCF-1018388). Work by L. Schlipf was supported by the Deutsche Forschungsgemeinschaft within the research training group "Methods for Discrete Structures"(GRK 1408). V. Polishchuk is funded by the Academy of Finland grant 138520.

4.1 Introduction

Let S be a finite set of line segments in the plane. We say that S is *stabbable* if there exists a convex polygon whose boundary $\mathcal{C}$ intersects every segment in S; the closed convex chain $\mathcal{C}$ is then called a (convex) *transversal* or *stabber* of S.

Research on transversals is an old and rich area. Most of the work, however, has focused on *line* transversals, i.e., on determining properties of families of *lines* that stab sets of various types of geometric objects. Stabbing has attracted interest from various perspectives: purely combinatorial (complexity of the set of transversals, orders induced by stabbers), algorithmic (computing the stabbers), and applied (using transversals in curve reconstruction, line simplification, graphics, motion planning) – see [86] and references therein. In some of these applications it is natural to consider convex transversals as generalizations of line transversals.

The problem of computing a convex transversal was posed in 1987 [164]. For the case of stabbing *vertical* line segments, an optimal algorithm for the problem was presented by Goodrich and Snoeyink in [68]. They stated the problem of finding a convex stabber for a set of *arbitrary* segments in the plane as open. To the best of our knowledge, there has been no progress on the problem in the roughly 20 years since then.

4.1.1 Contributions

We prove that finding a convex transversal for a set of segments in the plane is NP-hard; the problem remains NP-hard for a set of scaled copies of a given convex polygon. We also show that in 3D, it is NP-hard to decide stabbability of a set of balls.

We then turn to positive results: Section 4.3 presents a dynamic program (DP) to decide if a set of *pairwise-disjoint* segments is stabbable by a stabber whose vertices are a subset of a given candidate set of points; if the segments are not stabbable, we can output a convex stabber that intersects the maximum number of segments. The algorithm readily generalizes to the case of disjoint convex polygons. (In an earlier version of the chapter (see, e.g., [15]) we erroneously claimed that there always exists a stabber with edges supported by bitangents between elements of S. We also claimed that our algorithm extends directly to the case of convex pseudodisks; however, the details of that extension are not straightforward and will be the topic of a future follow-on paper.)

We also consider the *approximate symmetry detection* problem: Given a

set of n disks in the plane and an integer k, is it possible to find one point per disk such that the points form a set invariant under rotations by $2\pi/k$? For general k, the problem is NP-hard [85]; in Section 4.4 we give a polynomial-time algorithm for the case $k = n$. That is, we answer the question: is it possible to find one point per disk such that the points are vertices of a regular polygon? We also consider an optimization variant of the problem: Given a set of points in the plane, find the minimum δ^* such that shifting each point by at most δ^* brings the points into a symmetric position.

4.1.2 Closed stabbers vs. Terrains

The stabbing problem formulation is isotropic in the sense that it does not single out any specific direction in the space. In function approximation and statistics applications (unlike in surface reconstruction), it is often the case that the transversal represents the graph of a function. That is, the stabber is a *terrain* – a surface that intersects every vertical line in at most one point. A *convex* terrain is a part of the boundary of a convex polygon (polytope in 3D).

Finding a convex terrain stabber is a special case of finding a convex stabber – to see this, just place one point far below the input (Fig. 4.1). Our results, both positive and negative, are as strong as possible with respect to the distinction between convex terrain and convex stabbers: Our DP allows one to find even a convex stabber (and, hence, also to find a convex terrain stabber); our negative results show that it is hard already to find a convex terrain (and, hence, it is also hard to find a closed convex stabber).

4.2 Hardness results

This section gives an answer to the question from [68, 164] by showing that the problem is NP-hard.

4.2.1 Stabbing segments in the plane is NP-hard

Our reduction is from 3SAT. Our reduction is very similar to the one used to show hardness of finding the largest-area convex hull of a set of points that are restricted to lie on line segments [122]. The reduction is shown in Fig. 4.2. We use n and m to denote the number of the 3SAT variables and clauses, respectively.

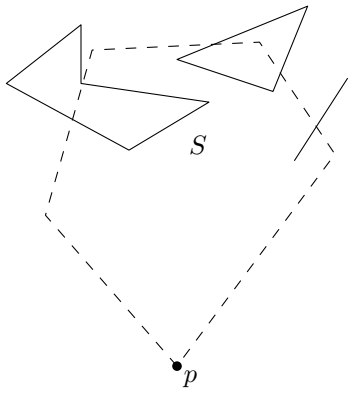


Figure 4.1: S' is S augmented with a point p. S can be stabbed by a convex terrain if and only if S' has a convex stabber. Thus, any algorithm that finds a stabber can also find a terrain. Conversely, if finding a terrain is hard, finding a stabber is also hard.

Variable gadget For each variable we have a gadget that consists of three points (segments of zero length) and one segment. There are two ways to traverse the gadget (shown with dotted and dashed paths) that differ in the order in which the middle point and the segment are visited. The two ways correspond to setting the variable True or False. The important property of the gadget is that it will be possible to place a certain "connecting" segment in either of the two ways: so that it touches only the False subpath but not the True, and vice versa.

"Squashing" We make the variable gadget "thin" by moving all three points close to the supporting line of the segment, and, in addition, by moving the non-middle points far apart.

Variable chain Variable gadgets are placed along a convex chain, called the *variable chain*. The chain is almost vertical, bending to the right only slightly. The variable gadgets are "clenched" onto the chain, and the distance between consecutive gadgets is large. Thus, the only way to traverse the gadgets with a convex terrain is to visit them one by one, in the order as they appear along the chain, assigning truth values to the variables in turn in each gadget.

Clause gadgets The clause gadgets are similarly arranged, one after one, on another almost vertical convex chain, slightly bending to the left; this clause

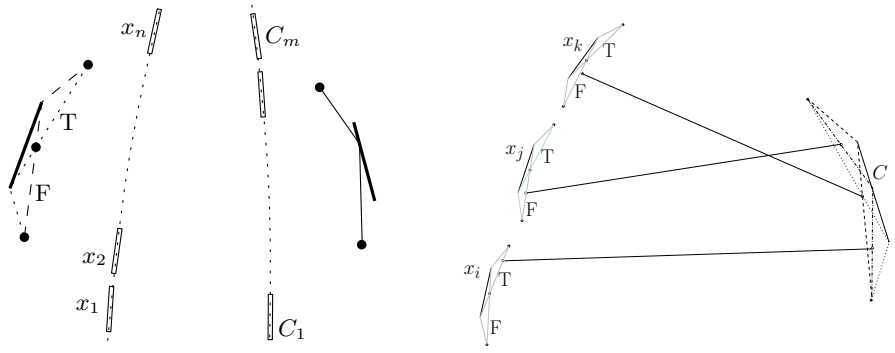


Figure 4.2: From left to right: The variable gadget and the two ways to traverse it. The variable gadgets are threaded onto a convex chain; similarly, the clause gadgets are threaded. The chains (dotted) are not parts of the construction and are shown only for reference. The clause gadget can be traversed in only one way. A clause $C = x_i \vee \bar{x_j} \vee \bar{x_k}$: three paths are shown that pick different subsets of the three connecting segments. The gadgets and their locations are not to scale: the gadgets are thinner, so that the points are very close to the supporting line of the segment – this makes the turn angles of the paths close to π; also, consecutive gadgets along each chain are separated so that a convex terrain can make independent choices in each of them.

chain is placed to the right of the variable chain. Each clause gadget consists of 2 points and a segment; the only way to traverse the gadget is to visit the first point, then the segment and then the second point – the only flexibility is where to touch the segment.

Connectors We now place $3m$ more segments, connecting a variable gadget to a clause gadget whenever the variable appears in the clause. The placement of the segments' endpoints within variable gadgets is as follows: if the variable appears unnegated, the segment touches the True path through the gadget and does not intersect the False subpath; on the contrary, if the variable appears negated, the segment touches only the False subpath. In every clause gadget, segments' endpoints look the same – see Fig. 4.2; as can be easily checked, a convex terrain can intersect any two of the segments, but not all three. This finishes the construction.

The reduction If the 3SAT instance is feasible, the stabber can traverse the variable gadgets according to the satisfying truth assignment. In each of

the clauses, at least one of the connecting segments (the one connecting to the satisfying variable) can be omitted; the other two are picked up by one of the three paths.

Conversely, if there exists a stabber, it must omit (at least) one connecting segment per clause. Set the variable True or False depending on whether the omitted segment connects from a True or False part of the variable gadget; this satisfies all the clauses. The True/False setting is consistent because any segment omitted by the stabber in the clause gadget must have been stabbed in the variable gadget, and there either only the True-subpath or only the False-subpath segments could have been stabbed, but not both.

We thus have our main negative result:

Theorem 4.2.1. *Finding a convex (terrain) transversal for a set of segments in the plane is NP-hard.*

In the remainder of this section we modify our proof to show hardness of stabbing scaled copies of a convex polygon (Section 4.2.2) and hardness of stabbing balls in 3D (Section 4.2.3).

4.2.2 Stabbing squares and scaled copies of a convex polygon

To show hardness of stabbing squares we again reduce from 3SAT. The construction (Fig. 4.3) is very similar to the one for segments.

Variable gadgets The variable gadget consists of three points (squares of zero area) and a square. There are two ways to traverse a gadget; one corresponds to setting the variable True and the other to setting the variable False.

"Fitting" We fit the variable gadget into a circular arc by putting the two non-middle points on the arc. The middle point and the lower edge of the square (the edge that is closest to the three points) lie inside the circular arc, see Fig. 4.3. Each variable gadget is fit into an arc of $1/(8n)$ of a unit circle.

Variable arc The variable gadgets are placed next to each other on an arc of one eighth of a unit circle. We call this arc the *variable arc*. The only way to traverse the variable gadgets with a convex terrain is to visit them one by one, in the order they appear on the arc, assigning truth values in turn in each gadget.

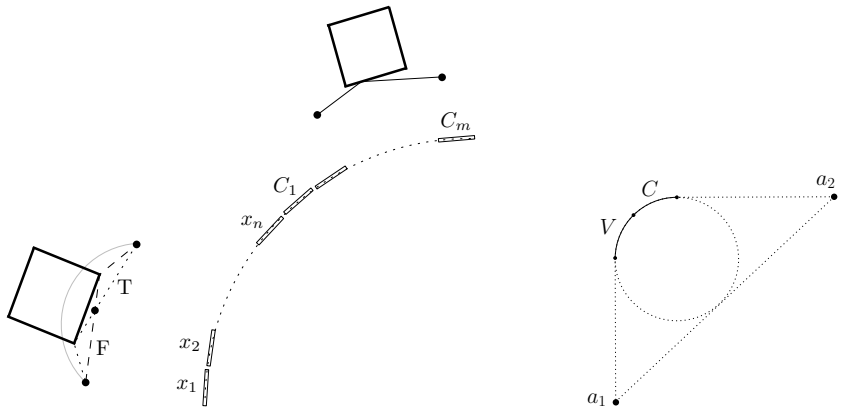


Figure 4.3: From left to right: The variable gadget and two ways to traverse it; the gadget is nearly the same as in the construction for segments, but instead of a segment we use a square (the same holds for the clause gadgets). The variable gadgets are placed on an arc of one eighth of a unit circle; the clause gadgets are placed also on an arc of one eighth of the unit circle, next to the variable gadgets. In total, the gadgets occupy an arc of one fourth of the circle. The gadgets are not to scale: each variable gadget is fit into the circular arc of length $1/(8n)$ and each clause gadget is fit into the circular arc of length $1/(8m)$; also consecutive gadgets are separated so that a convex terrain can make independent choices in each gadget. On the right, V and C mark the placements for the variable and the clause gadgets respectively; the points a_1, a_2 ensure that the connector squares can either be intersected at a variable gadget or a clause gadget but nowhere else.

Clause gadgets The clause gadgets are placed in the same way as the variable gadgets, on an arc of one eighth of the unit circle and next to the variable arc. Each clause gadget consists of two points and a square.

Connectors We place $3m$ more squares, connecting a variable gadget to a clause gadget whenever the variable appears in the clause (Fig. 4.4). One edge of each square is placed exactly in the same way as the connector segment in the construction for line segments. This means that one endpoint of the edge lies within the variable gadget as follows: if the variable appears unnegated, the edge touches the True subpath through the gadget and does not intersect the False subpath; on the contrary, if the variable appears negated, the edge touches only the False subpath. In every clause gadget, the endpoints of these edges look the same.

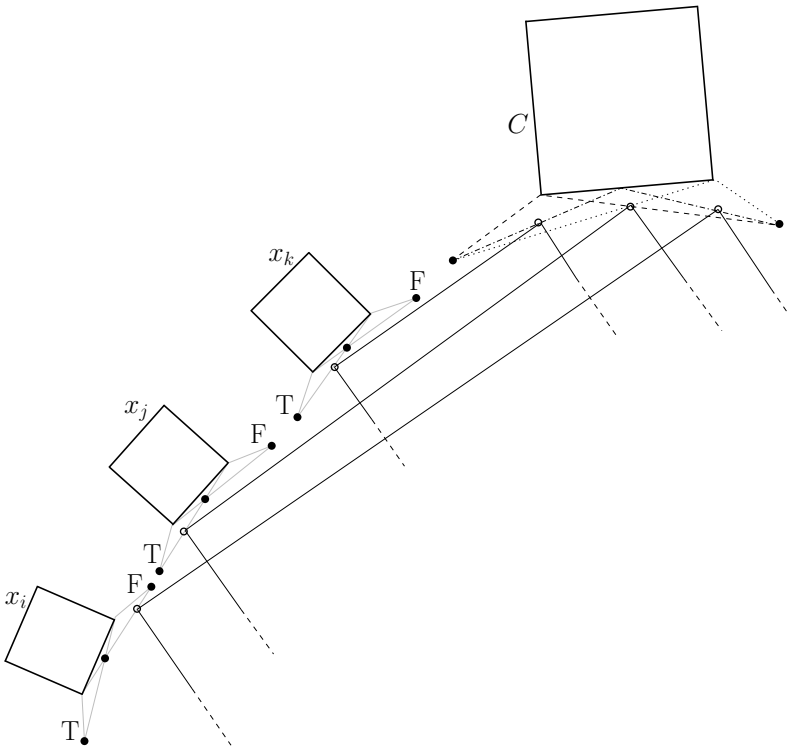


Figure 4.4: Clause $C = x_i \vee \bar{x_j} \vee \bar{x_k}$. Three path are marked that pick up different subsets of the three connecting squares.

To ensure that a convex terrain can intersect connecting squares only near the gadgets, we add two points to the construction (Fig. 4.3, right); a convex terrain that traverses these points and all gadgets cannot intersect the unit circle (on which the gadgets are placed) except at the gadgets. Thus, the connectors can be intersected only at endpoints of the edges that are placed in the same way as the connector segments in the construction for line segments. (Note that for the latter property to hold, it was crucial to fit all variable and clause gadgets on the quarter of a unit circle – this way all connector squares lie inside the unit circle.)

The reduction Assume that the 3SAT formula is feasible. Then the stabber can traverse the variable gadgets according to the satisfying assignment. In each clause gadget one of the three connecting squares has to be omitted by the stabber; let this be the one connecting to the satisfying variable.

On the other hand, if there exists a stabber, it must omit at least one

connecting square per clause. Set the variables True or False depending on whether the omitted square connects from a True or False path of the variable gadget; this satisfies all the clauses. This setting is consistent since any square omitted by the stabber in the clause gadget has to be stabbed in the variable gadget and there either only the True-subpath or the False-subpath squares could have been traversed, but not both.

Theorem 4.2.2. *Finding a convex (terrain) transversal for a set of squares in the plane is NP-hard.*

Generalization The above proof can be adapted to show that stabbing regular k-gons is NP-hard for any $k > 2$: just replace the squares with the k-gons, and (to ensure again that the connectors lie inside the unit circle) place the variable and clause gadgets on an arc of $1/(2k)$ of a unit circle. That is, fit each variable gadget into an arc of $1/(4kn)$, and each clause gadget into an arc of $1/(4km)$.

Theorem 4.2.3. *For arbitrary $k > 2$, finding a convex (terrain) transversal for a set of regular k-gons in the plane is NP-hard.*

It is not crucial that the polygons are regular, as only one edge of each polygon is important for the construction. Hence, the construction works in the same way if we consider a set of scaled copied of a given polygon instead of regular polygons.

Theorem 4.2.4. *Finding a convex (terrain) transversal for a set of scaled copies of a given convex polygon in the plane is NP-hard.*

An interesting open question is whether stabbing disks is NP-hard. Our reduction above does not extend to this case – the reason is that we want the connectors to lie inside the unit disk onto which the variable and clause gadgets are threaded (this is needed to ensure that the connectors cannot be stabbed outside the unit circle – the only places to stab them are near the gadgets).

4.2.3 Stabbing balls in 3D is NP-hard

We again reduce from 3SAT, employing similar ideas as those for segments in 2D.

(a) Variables

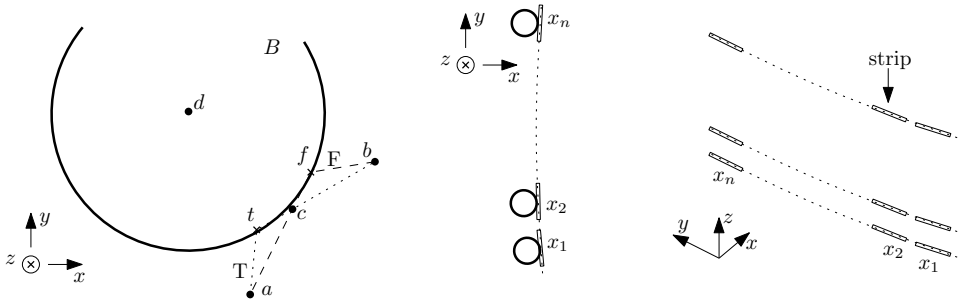


Figure 4.5: Left: The (cross-section by the supporting plane of the) variable gadget. The gadget is not to scale; actually, the turn angles of both the dashed and the dotted paths are close to π. Middle: The variable gadgets are threaded onto a convex chain; the chain (dotted) is not a part of the construction and is shown only for reference. Right: The variable grid.

Variable gadget The basic variable gadget consists of three points a, b, c (balls of 0 radius) and one ball B of large radius whose center is denoted by d (Fig. 4.5, left). The three points and the center of the ball all belong to a horizontal plane, which we call the *supporting plane* of the gadget.

True and False touches The cross-section of B by the supporting plane is a disk. The points t, f where the tangents from a and b touch the disk are called the True and the False *touches*.

"Squashing" As with the segments in 2D, we make the dashed and dotted paths (see Fig. 4.5, left) have the turn angles close to π. For that, we make the three points a, b, c almost collinear, moving a and b far apart, and using a large radius for the ball B.

Variable chain Also as with segments in 2D, the variable gadgets are placed along an almost "flat" convex chain (Fig. 4.5, middle). Again, the gadgets are "clenched" onto the chain so that the three points of every gadget are very close to the chain. All gadgets and the chain are aligned, in that the supporting planes of all gadgets coincide, and the chain also lives in this common horizontal plane. We thus also call the plane the *supporting plane* of the chain. The balls are "sticking out" of the chain, i.e., the centers of the balls are placed outside the convex hull of the chain.

As with segments in 2D, consecutive gadgets along the chain are separated by

large enough distance so that the cross-section of the stabber by the supporting plane must visit the gadgets one by one, assigning truth values to the variables in turn in each gadget. We call this whole construction—the gadgets threaded on the chain—the *variable chain*.

Variable grid We place $m+2$ copies of the variable chain, one copy directly above another (Fig. 4.5, right). We number the copies from 0 to $m+1$. The first and the last copies are "dummy"; we have them only to enforce consistency of the "choices" that the stabber must make in each of the chains (see below). The other copies correspond to the clauses.

We call the $m+2$ gadgets corresponding to variable x_i in all $m+2$ chains the i-th *variable strip*. This way, our construction so far is a "grid" of n strips $\times\ m+2$ chains (see also Fig. 4.7 below).

Consistency In Section 4.2.3 we argue that any convex terrain must make the same "choices" at each of the m copies of the variable in a strip. That is, for all $j = 1, \ldots, m$, in the cross-section by the supporting plane of the jth chain, the stabber either uses the True touch or uses the False touch.

(b) Clauses

Clause chains Place $2(2m+2)$ points (0-radius balls) on two identical parallel almost vertical convex chains P, Q, slightly bending to the left, lying in planes that are perpendicular to the y-axis (Fig. 4.6, left). More specifically, the first two points p_0, p_1^- of P are the endpoints of a segment parallel to the z-axis. The next two points, p_1^+, p_2^- are the endpoints of a segment making a slightly larger than 0 angle with the z-axis. The next two points, p_2^+, p_3^- are endpoints of a segment making even larger angle with the z-axis, and so on: the segments $p_j^+ p_{j+1}^-$ make larger and larger angles with the z-axis as $j = 1, \ldots, m-1$ increases. The last two points of P are p_m^+, p_{m+1}. That is, P has 2 points per clause, plus the points p_0, p_{m+1}.

The chain Q is a parallel shift of P in the y direction. The points belonging to Q are analogously numbered $q_0, q_1^-, \ldots, q_m^+, q_{m+1}$. We place P and Q to the right of the variable grid (see Fig. 4.7, left).

Plates Because P and Q are parallel to each other, the quadruple of points p_0, p_1^-, q_1^-, q_0 lie in the same (vertical) plane parallel to the y-axis; we call

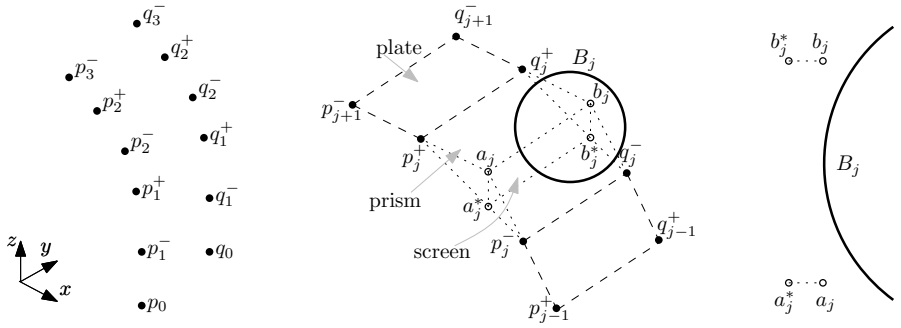


Figure 4.6: Left: The chains P, Q. Middle: The plates (dashed), and the prism and the screen (dotted). The figures are not to scale; actually, the chains are almost vertical (and the supporting planes of screens are almost horizontal). Right: The cross-section of the gadget for clause j by the supporting plane of the jth screen.

the rectangle $p_0p_1^-q_1^-q_0$ the *0th plate*. Similarly, $p_1^+, p_2^-, q_2^-, q_1^+$ lie in the same plane (making a positive angle with the z-axis) also parallel to the y-axis; the rectangle $p_1^+p_2^-q_2^-q_1^+$ is the *1st plate*. In general, points $p_{j-1}^+, p_j^-, q_j^-, q_{j-1}^+$ lie in a plane parallel to the y-axis, making larger angle with the z-axis for larger j; the rectangle $p_{j-1}^+p_j^-q_j^-q_{j-1}^+$ is the *jth plate* (Fig. 4.6, middle).

Clause gadgets By construction, the plates must be inside the convex hull of the stabber (including the possibility of some plates being part of the stabber itself). Extend the plates by sliding out the sides $p_j^+q_j^+, p_j^-q_j^-$ until the sides intersect along a segment a_jb_j. The points $p_j^+, q_j^+, p_j^-, q_j^-, a_j, b_j$ define a right triangular prism (with bases perpendicular to the y-axis), which we call the *jth prism*. (Note that the triangles in the prism base are not right; it is the prism that is right.)

The *jth screen* is the rectangle $a_jb_jb_j^*a_j^*$ where $a_ja_j^*, b_jb_j^*$ are altitudes of the triangles $p_j^-p_j^+a_j$, $q_j^-q_j^+b_j$, respectively. The gadget for the clause j is a ball B_j with the center in the supporting plane of the screen. The ball intersects a_jb_j but does not intersect $a_j^*b_j^*$ (Fig. 4.6, right). The exact placement of B_j depends on which variables constitute the clause j, as detailed below in Section 4.2.3.

(c) The reduction

The points and balls described so far can be stabbed by a convex (surface) stabber: the stabber can traverse the variable grid making arbitrary (but

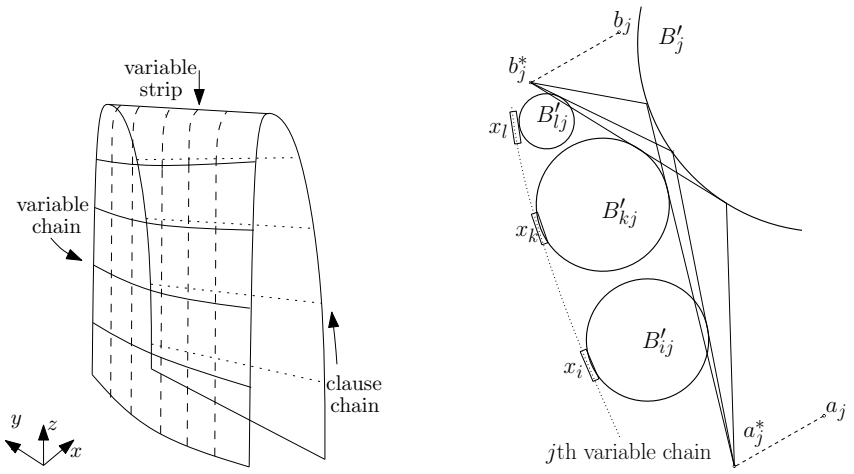


Figure 4.7: Left: The "frame" of the construction. Right: The cross-section by the supporting plane of the jth screen.

consistent, across the chains) truth assignments for the variables, and then turn onto the P, Q-side where the stabber can use plates and prisms. We now place $3m$ more balls, connecting a variable gadget to a clause gadget whenever the variable appears in the clause; this turns our instance into one that has a stabber if and only if there exists a satisfying assignment in the 3SAT.

Tall, narrow and deep First of all, we align the variable grid and clause chains P, Q so that each pair (jth chain, jth clause gadget) lives in its own horizontal slab, called the *jth slab*. We fine-tune the angles of P, Q so that the supporting plane of the jth screen almost coincides with the supporting plane of the jth variable chain (they cannot coincide fully because the supporting plane of the variable chain is horizontal, while that of the screen is not). Next, we make the whole construction "tall" and "narrow" so that the distance between the jth chain and the jth clause gadget is smaller than the height of the jth slab. We also make the construction "deep" in the y direction: the y-span of a variable chain, as well as the distance between P and Q, is large in comparison to the distance between the jth variable chain and the jth clause gadget, for any j. Refer to Fig. 4.7, left.

Connectors Suppose now that the jth clause contains variables $x_i, x_k, x_l, i < k < l$. We place three *connecting balls* B_{ij}, B_{kj}, B_{lj} with the centers lying in the supporting plane of the jth screen (Fig. 4.7, right). The balls are placed

opposite the variable gadgets for x_i, x_k, x_l in the jth variable chain, and each ball spans the space inside the construction between the jth variable chain and the jth clause gadget (the height, the narrowness and the depth of the construction allows us to place the balls so that they are disjoint from the analogous balls in the other slabs).

Similarly to the case of segments in 2D, if x_i is unnegated in clause j, the ball B_{ij} touches the True (dotted, in Fig. 4.5, left) path through the x_i's gadget in the jth variable chain and does not touch the False (dashed) path. Otherwise, the ball touches the False path and not the True path. The placement of B_{kj}, B_{lj} is analogous. The interaction of the balls with the jth clause gadget is also similar to the segments in 2D: we place the balls so that there exists a convex terrain intersecting any two of the three balls, but not all three (Fig. 4.7, right). Section 4.2.3 details how to do this.

Correctness If the 3SAT instance is feasible, then the stabber can traverse the variables gadgets according to the satisfying truth assignment. In each of the clauses, the ball connecting to the satisfying variable is omitted by the stabber; the other two are picked up in the clause gadget.

Conversely, if there exists a stabber, it must consistently traverse the variable gadgets in the variable grid setting the truth assignment. On the clauses side, the stabber must contain (inside its convex hull) all plates. "In between" the plates (i.e., inside the prisms) the stabber is free to do whatever it likes; however, no matter how it goes it will not be able to stab more than two connecting balls per clause. The unstabbed ball satisfies the clause; the consistency of the satisfying assignment follows from the fact that a variable cannot be set both to True and to False by the same stabber.

Precision We were informal in saying that parts of the construction are "large" enough, "far" enough, etc. Still, the equations and inequalities involving the coordinates of the points in the gadgets have polynomial-size coefficients. E.g., a variable (resp. clause) chain can be part of the boundary of the regular $O(n)$-gon (resp. $O(m)$-gon). Thus, the construction can be done so that it has the required properties and the coordinates specifying positions of the parts of the gadgets are polynomial in n and m.

Overall, we have:

Theorem 4.2.5. *Finding a convex (terrain) transversal for a set of balls in 3D is NP-hard.*

(d) Consistency of choices in a variable strip

Let T be a convex terrain stabber, and consider a fixed i. We claim that the cross-section of T by the supporting plane of the jth chain looks the same for all $j = 1, \ldots, m$ in the vicinity of the variable gadget for x_i: the stabber either uses the True touch or uses the False touch of the gadget. The proof is based on the following straightforward observations:

Lemma 4.2.6. *Let C be the convex hull of T. Consider any basic variable gadget for x_i (Fig. 4.5, left). We have:*

i. Either the True or the False touch belongs to C, but not both.

ii. No point of the segment cd other than c belongs to C; i.e., $cd \cap \mathcal{C} = \rfloor$.

Say that the stabber makes a *switch* if it sets x_i True in the jth variable chain but sets x_i False in the $j+1$st chain, or vice versa, for some $j = 1, \ldots, m$. Consider the two cases:

There is more than one switch. Without loss of generality suppose that x_i is set to True in chains j_-, j_+ and to False in a chain j, for $j_- < j < j_+$. Let t_-, t, t_+ be True touches in x_i's gadget in the chains j_-, j_+; let f be the False touch in the chain j (Fig. 4.8, left). We know that $t_-, t_+, f \in \mathcal{C}$. The True touches of x_i's gadgets in all chains lie on a common line, i.e., t is a point on the segment t_-t_+. Thus, since C is convex, $t \in \mathcal{C}$. This, together with $f \in \mathcal{C}$ contradicts Lemma 4.2.6i.

There is exactly one switch. Without loss of generality suppose that x_i is set to True in a chain j and to False in the chain $j+1$. Since $j \geq 1$, there exists chain $j-1$. If x_i is set to False in it, then there is more than one switch. Otherwise, let t_- be the True touch in $j-1$st chain and let f^+ be the False touch in $j+1$st chain (Fig. 4.8, right); let h be the intersection of the segment t_-f_+ with the supporting plane of the jth chain. By symmetry, $h \in cd$ where c and d are the middle point and the center of the large ball in the jth gadget for x_i (refer to Fig. 4.5, left). Since $t_-, f_+ \in \mathcal{C}$ and C is convex, $h \in \mathcal{C}$. This, together with $h \in cd, h \neq c$ contradicts Lemma 4.2.6ii.

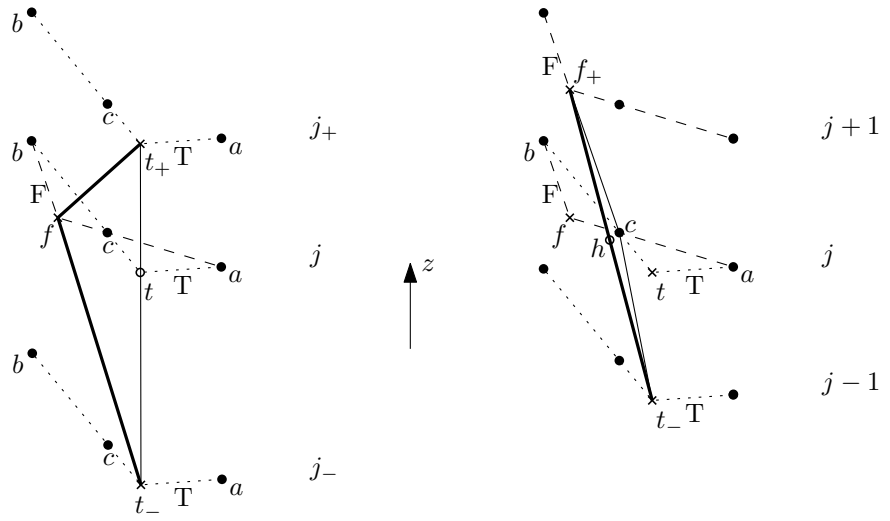


Figure 4.8: Left: If f is in the stabber, then t is not; however if t_-, t_+ are in, then t must be in too. Right: If t_-, f_+ are in the stabber, then a point $h \neq c$ of the segment cd is in the stabber.

Two, but not three, connecting balls can be stabbed in a clause gadget

We show how to place the three connecting balls and the ball of the jth clause gadget so that any two balls can be stabbed by a convex terrain, but all three cannot.

At most two balls are stabbed. First of all, each of the segments $a_j a_j^*, b_j b_j^*$ (sides of the jth screen) must have a point of the stabber in it. Denote by $B'_{ij}, B'_{kj}, B'_{lj}, B'_j$ the cross-sections of the balls $B_{ij}, B_{kj}, B_{lj}, B_j$ by the supporting plane of the jth screen (Fig. 4.9). Let $a_j^* a_k, b_j^* b_k$ be the rays from a_j^*, b_j^* tangent to B'_{kj}. Choose the radius of B'_{kj} so that the intersection point of the tangents is close to $a_j b_j$ (but is still inside the screen rectangle $a_j b_j b_j^* a_j^*$).

Increase the radius of B_{ij} from 0 just past the value at which B'_{ij} is tangent to $a_j^* a_k$ (i.e., the ray $a_j^* a_k$ cuts off a positive-area cup from the disk B'_{ij}). Choose the radius of B'_{lj} similarly. Now draw the tangent $a_j^* a_i$ from a_j^* to B'_{ij} and the tangent $b_j^* b_l$ from b_j^* to B'_{lj}; let r_k be the intersection of the tangents. We place the ball B_j so that B'_j goes through r_k and is tangent to the rays $a_j^* a_k, b_j^* b_k$ (the tangency points are denoted r_i, r_l). It easy to see that no convex terrain can stab all 3 balls B_{ij}, B_{kj}, B_{lj} provided it

stabs B_j.

Stabbing two balls. On the other hand, any two of the balls can be intersected by a convex stabber. For that, one can use a *barn roof* (Fig. 4.10) which is a construction, with 4 faces, fully lying inside the jth prism (so as not to break the overall convexity of the stabber). The two opposite faces of the roof are congruent triangles $p_j^- q_j^- s_j, p_j^+ q_j^- t_j$ attached to the adjacent plates coplanarly with the plates. The segment $s_j t_j$ is parallel to $p_j^- p_j^+$ (and $q_j^- q_j^+$). The segment can be shifted between $p_j^- p_j^+$ and $q_j^- q_j^+$ arbitrarily so that the cross-section of the roof by the supporting plane of the jth screen looks like either of the 3 paths a_j^*-r_i-b_j^*, a_j^*-r_k-b_j^*, a_j^*-r_l-b_j^* in Fig. 4.9. That is, the roof can intersect any two of the balls B_{ij}, B_{kj}, B_{lj} in the clause gadget (Fig. 4.10, right).

4.3 Stabbing disjoint segments

This section presents a dynamic program (DP) to decide stabbability of a set S of *pairwise-disjoint* segments in the plane by a convex stabber whose vertices are restricted to come from a given discrete set $C \subset \mathcal{R}^\in$ of candidate points. A subproblem in the DP is specified by a pair of potential stabber edges together with a constant-complexity "bridge" between the edges (the bridge is either a single segment or a segment—visibility-edge—segment chain). The disjointness of the segments allows us to determine which segments must be stabbed within the subproblem. We show that a segment-free triangle can be found that separates a subproblem into smaller subproblems, which allows the DP to recurse.

Arcs and nodes, chords and bridges A straight-line segment between two points from C (i.e., a potential stabber edge) is called an *arc*. Two arcs pq, rt are *compatible* if either they have a common endpoint or the supporting lines of the arcs intersect outside each of pq, rt. In other words, the points p, q, r, t are in convex position, and pq, rt have the potential to be sides of a convex polygon – the stabber. Refer to Fig. 4.11, left.

Let $\mathcal{P}'$ denote the set of points at which arcs intersect segments from S. Let $\mathcal{P}$ be the union of $\mathcal{P}$', C, and endpoints of segments from S; call points in $\mathcal{P}$ *nodes*. A *chord* is a straight-line segment whose interior intersects no segment from S, and whose each endpoint is a node.

A *bridge* is a polygonal path with the following properties:

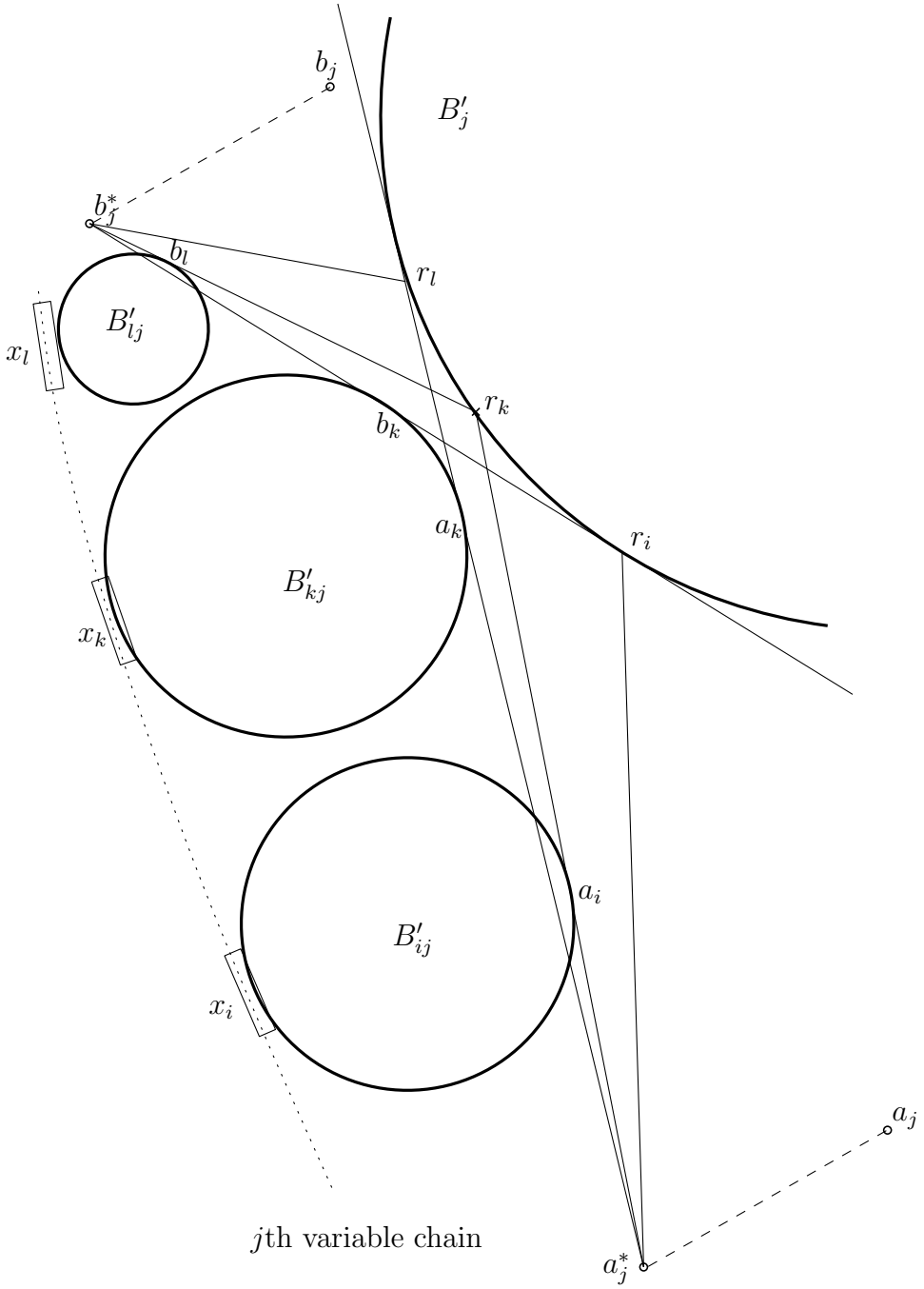


Figure 4.9: The cross-section by the supporting plane of the jth screen.

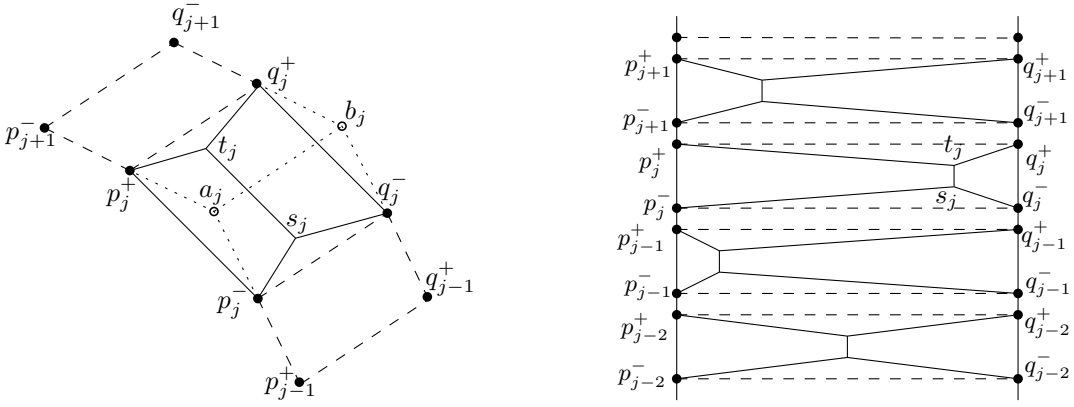


Figure 4.10: Left: The roof. s_jt_j is below a_jb_j. s_j (resp. t_j) lies in the plane of the plate $p^+_{j-1}p^-_jq^-_jq^+_{j-1}$ (resp. $p^-_{j+1}p^+_jq^+_jq^-_{j+1}$). Right: View of the clause-side of the stabber from a point at $+\infty$ on the x-axis; the stabber edges are bold, the plates boundaries are dashed. s_jt_j can be shifted freely to grab any two of the balls B_{ij}, B_{kj}, B_{lj} as in Fig. 4.9; similarly, the ridges of the roofs can be shifted independently within each clause gadget.

- Its endpoints are nodes.
- It has at most three links.
- (i) If it has exactly one link, then the link is either a chord or a part of a segment from S – in the latter case, the bridge is *chordless*; (ii) if it has exactly two links, then one of the links is a chord, and the other is a part of a segment; (iii) if it has exactly three links, then they are a part of a segment from a node to the segment endpoint, a chord, and a part of another segment from the segment endpoint to a node (that is, the chord connects endpoints of the two segments).

Subproblems A subproblem in our DP is specified by two compatible arcs and a bridge. More specifically, let $p, q, r, t \in C$ be the four candidate points forming compatible arcs pq, rt. Without loss of generality let rt be below the line pq, and let q, p, r, t be the order in which the nodes appear counterclockwise on the convex hull of the arcs. We define the wedge W to be the region that is below the line supporting pq and above the line supporting rt. In addition to the two arcs, the subproblem has in the input a bridge B that connects some point of pq to some point of rt. Refer to Fig. 4.11, middle.

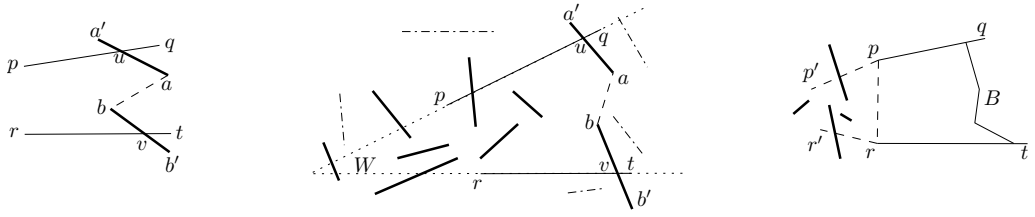


Figure 4.11: Left: $p, q, r, t \in C, aa', bb' \in S, u, v \in \mathcal{P}', p, q, r, t, a, a', b, b', u, v \in \mathcal{P}$. pq, rt are compatible arcs. $rp, ba, bu, av, ta, aq, tu, bp, rb, ra, ap$ are chords. $rp, vbau, vau, vbu, vbp$ are some of the bridges; rp is chordless. Middle: The wedge W (boundaries dashed) and the bridge $B = vbau$. The segments in $S_{pq,rt,B}$ that have to be stabbed to the left of B are solid; the segments in $S \setminus S_{pq,rt,B}$ are dash-dotted. Right: An empty subproblem (pq, rt, B) and an induced subproblem $(p'p, r'r, rp)$.

Subproblem's responsibility The crucial observation that allows us to run the DP is the following: Assuming that the arcs pq, rt are part of the stabber, we know for each segment $s \in S$ whether it should be stabbed to the left or to the right of the bridge. Indeed, only those segments that have non-empty intersection with the wedge W can be stabbed. On the other hand, no segment can have points on both sides of the bridge – for that it would have to cross the bridge, and this is impossible: the chord is not crossed by definition, and no segment is crossed by another segment due to the assumption of pairwise-disjointness of segments in S.

Let $S_{pq,rt,B}$ denote the segments that must be stabbed to the left of the bridge B; i.e., the segments that intersect W in the part of the wedge that lies to the left of B.

The function Stab$(\cdot)$ Define a Boolean function Stab(pq, rt, B) to be True if the segments $S_{pq,rt,B}$ can be stabbed (assuming pq, rt is a part of the stabber), and to be False otherwise; for an incompatible pair of arcs pq, rt define Stab$(pq, rt, \cdot)$ to be always False. The function shows whether the stabber can be "completed" having pq, rt as its part. In the remainder of this section we show how to evaluate the function on a subproblem given its values at other subproblems, i.e., how to solve the DP.

Empty subproblems The subproblem (pq, rt, B) is *empty* (Fig. 4.11, right) if no segment from S penetrates the region of W that is to the left of the bridge but to the right of rp (this includes the possibility that the bridge is the segment

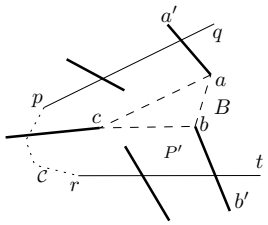


Figure 4.12: The (unknown) part of the stabber $\mathcal{C}$ is dotted. P' is the simple polygon bounded by the unknown part of $\mathcal{C}$, by pq, rt, by the bridge $B = tbaq$, and by the piercing segments. abc is a separating, i.e., segment-free triangle inside P'.

rp itself). An empty subproblem is *closed* if $p = r$. Closed subproblems are at the lowest level of our DP: clearly, $\text{Stab}(\sigma) = \text{True}$ for a closed subproblem σ.

Let (pq, rt, B) be an empty subproblem. We say that a subproblem $(p'p, r'r, rp)$ is an *induced* subproblem of (pq, rt, B) if pp' is below (the supporting line of) pq, and rr' is above rt. That is, the angles qpp' and trr' are convex, and thus both qpp' and trr' can potentially be parts of a convex chain – the stabber-to-be. Empty subproblems are easy to reduce to induced subproblems: $\text{Stab}(pq, rt, B) = \text{True}$ for an empty subproblem (pq, rt, B) if and only if $\text{Stab}(p'p, r'r, B)$ is True for at least one subproblem induced by (pq, rt, B).

General subproblems Let $\mathcal{C}$ be the sought stabber that has pq, rt as two of the sides (Fig. 4.12). (Of course, we do not know $\mathcal{C}$, but we will not use its existence in the algorithm, we will only use $\mathcal{C}$ to argue that we can split the subproblem into smaller ones.) Let $\mathcal{C}$' be the (convex) region bounded by $\mathcal{C}$, and let P be the part of $\mathcal{C}$' to the left of the bridge B (i.e., P is what is chopped off $\mathcal{C}$' by B). Consider the set $P' = P \setminus \bigcup_{s \in S_{pq,rt,B}} s$. That is, P' is P "pierced" by the segments $S_{pq,rt,B}$ that are stabbed in the subproblem (pq, rt, B).

Because $\mathcal{C}$ is a stabber, every segment in $S_{pq,rt,B}$ intersects the boundary of P. This means that P' is a (weakly) simple polygon (i.e., no segment makes a hole in P' by being fully contained in the interior of P'). Each vertex of P' belongs to one of the following 5 (overlapping) sets:

P_0**:** p, q, r, t

P_1**:** vertices of the bridge;

P_2**:** nodes that reside on the arcs pq, rt;

P_3**:** nodes that belong to $\mathcal{C}$ except those in P_2;

P_4**:** endpoints of segments from $S_{pq,rt,B}$ that are stabbed by pq or rt;

P_5**:** endpoints of segments from $S_{pq,rt,B}$ that are stabbed by $\mathcal{C} \setminus \sqrt{\text{II}}, \nabla\sqcup$.

Note that only P_3 is not known to us (because we do not know $\mathcal{C}$); all the other sets are known as soon as the subproblem (pq, rt, B) is specified.

We define the *important* link ba of the bridge B as follows: if B is chordless, then $ba = B$; otherwise ba is the chord of B. We assume that a is closer to pq, and b is closer to rt along B. Our algorithm will search for a separating, i.e., segment-free triangle abc within P' where c is a vertex of P' *and* $c \notin P_3$. We first argue that such a triangle exists, and next describe what to do depending on the set, among P_1, P_2, P_4, P_5, to which c belongs.

Lemma 4.3.1. *There exists a vertex c of P' such that $c \notin P_3$ and no segment intersects the interior of abc.*

Proof. The link ba is a side of P'; thus, any triangulation of P' has a triangle abc, with c being a vertex of P'. If there exists a triangulation such that $c \notin P_3$, we are done. Otherwise, let xy be the segment that contains c; i.e., $c = xy \cap \mathcal{C}$ (Fig. 4.13). Move c along xy inside P'. Either c reaches the endpoint of the segment (in which case we are done because $c \in P_5$) or one of the sides of abc, say, bc hits an endpoint z of a segment from $S_{pq,rt,B}$; let c' be the position of c on xy when this happens. The convex quadrilateral $cc'ba$ has no segments in the interior, and abz is the sought triangle. □

We emphasize that even though we used $\mathcal{C}$ in arguing the existence of the vertex as in the above lemma, we can find such a vertex without knowing $\mathcal{C}$ (e.g., just by trying all vertices in P_1, P_2, P_4, P_5).

Let $B = vbau$ be the bridge. We now show how our DP recurses into subproblems defined by the sides of the triangle abc (Fig. 4.14):

Case I: c is a vertex of the bridge; $c \in P_1$. Then the bridge has one fewer links, and $\text{Stab}(pq, rt, B) = \text{Stab}(pq, rt, B')$ where B' is the new bridge.

Case II: c is on pq, rt; $c \in P_2$. Without loss of generality suppose that $c \in rt$. If there exists a segment $s \in S_{pq,rt,B}$ that lies in the interior of the triangle vbc (i.e., s is not stabbed by tc), then s cannot be stabbed in the subproblem, and, hence, $\text{Stab}(pq, rt, B) = \text{False}$. Otherwise (i.e., if no segment

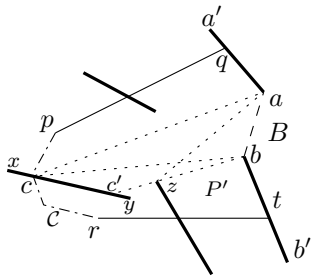


Figure 4.13: abc is a triangle in a triangulation of P'; move c inside P'. abz is the sought triangle.

intersects vbc or any segment that intersects tbc is already stabbed by rt), $\text{Stab}(pq, rt, B) = \text{Stab}(pq, rt, cau)$.

Case III: c is an endpoint of a segment from $S_{pq,rt,B}$ stabbed by pq, rt; $c \in P_4$. Without loss of generality suppose that c is the endpoint of a segment that is stabbed by rt; let z be the point of the stabbing. If there exists a segment $s \in S_{pq,rt,B}$ that lies in the interior of the quadrilateral $vbcz$ (i.e., s is not stabbed by tz), then s cannot be stabbed in the subproblem, and, hence, $\text{Stab}(pq, rt, B) = \text{False}$. Otherwise, $\text{Stab}(pq, rt, B) = \text{Stab}(pq, rt, zcau)$.

Case IV: c is an endpoint of a segment from $S_{pq,rt,B}$ stabbed by $\mathcal{C} \setminus \sqrt{\amalg}, \nabla\sqcup$; $c \in P_5$. Let d be the other endpoint of the segment touched by the triangle abc. Then $\text{Stab}(pq, rt, B) = \text{True}$ if and only if there exists an arc xy that intersects dc (say, at a point z) such that both $\text{Stab}(pq, xy, zcau)$ and $\text{Stab}(yx, rt, vbcz)$ are true. Formally,

$$\text{Stab}(pq, rt, B) = \bigvee_{\substack{\text{arc } xy: \\ dc \cap xy = z \neq \emptyset}} \left(\text{Stab}(pq, xy, zcau) \wedge \text{Stab}(yx, rt, vbcz) \right)$$

Extensions Our DP can be modified straightforwardly to find a convex stabber that stabs as many segments as possible. For that, we let the function $\text{Stab}(pq, rt, B)$ denote the number of elements of S stabbed by pq, rt plus the maximum number of other segments that can be stabbed in the subproblem (pq, rt, B). The recursions for the function change to reflect that $\text{Stab}(pr, qt, B)$ is the sum of the values of the function on the subproblems. Further, our DP extends immediately to solve the convex stabbing problem for disjoint convex polygons.

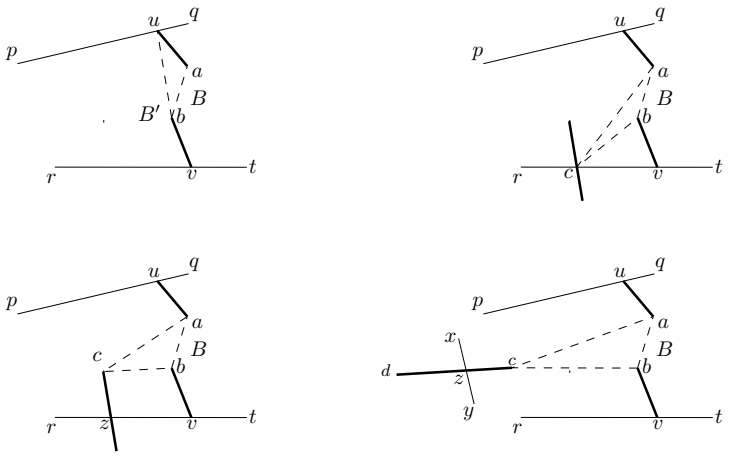


Figure 4.14: The DP recursion. Top: $c \in P_1, c \in P_2$. Bottom: $c \in P_4, c \in P_5$.

4.4 Stabbing with vertices of a regular polygon

In this section we present an algorithm to decide whether a given set of disks can be stabbed by a regular polygon. Specifically, the approximate symmetry detection problem is: Given a set of n disks in the plane and an integer k, is it possible to find one point per disk such that the points form a set invariant under rotations by $2\pi/k$? While the problem is NP-hard for general k [85], we solve the case $k = n$, i.e., we determine whether it is possible to find one point per disk so that the points are vertices of a regular n-gon.

4.4.1 The decision problem

Let $D = \{d_1, \ldots, d_n\}$ be the given disks. For points $p, c \in \mathcal{R}^{\in}$ and integer k let $\rho_c^k(p)$ denote the image of p after rotation around c by the angle $k2\pi/n$. For a pair of disks $d_i, d_j \in D$, let $A_{ij}^k = \{(p,c) | c \in \mathcal{R}^{\in}, \checkmark \in \lceil_{\rangle}, \rho_{\rfloor}^{\parallel}(\checkmark) \in \lceil_{|}\} \subset \mathcal{R}^{\triangle}$ be the set of all pairs (p,c) of points $p \in d_i, c \in \mathcal{R}^{\in}$ such that p moves to d_j after rotating by $k2\pi/n$ around c; we call A_{ij}^k the *apex region*.

Fix a disk d_1. A regular n-gon with a vertex per disk of D exists if and only if there exist $p \in d_1$ and $c \in \mathcal{R}^{\in}$ (the center of the n-gon) such that $\rho_c^j(p) \in d_{j+1}$ for $j = 1, \ldots, n-1$, or in other words, if and only if the intersection of $n-1$ apex regions A_{1j+1}^j is non-empty (here the vertices of the regular n-gon stab the disks in the order $d_1, d_2, \ldots$; of course this order is not known in advance).

This prompts us to go through "all possible" intersections between the apex regions, checking for each of the intersections whether an n-gon exists.

Specifically, consider the $(n-1)^2$ apex regions $A^k_{1,j}, j = 2, \ldots, n, k = 1, \ldots, n-1$. Call a point $(p, c) \in \mathcal{R}^{\triangle}$ *feasible* if it belongs to some $n-1$ of the regions, with each region being from a different disk with a different angle. Our problem has a feasible solution if and only if there exists a feasible point in $\mathcal{R}^{\triangle}$.

There are $O(n^2)$ apex regions, and each is defined by 2 polynomials of constant degree; thus, the arrangement of the regions has polynomial complexity. The feasibility of a point in $\mathcal{R}^{\triangle}$ does not change as the point moves inside the cell of the arrangement; hence, in order to determine existence of a feasible point, it is enough to check the feasibility of an arbitrary representative point $r = (p, c)$ inside every cell. By [19], a representative for each cell can be obtained in $O(n^2)$ time.

To check if $r = (p, c)$ is feasible, build the bipartite graph G_r; the $n-1$ nodes on one part correspond to the disks $D \setminus d_1$, the $n-1$ nodes on the other part correspond to the angles $\{\pi/n, 4\pi/n, 6\pi/n, \ldots, (n-1)2\pi/n\}$. There is an edge between a disk node d_j and an angle node $k2\pi/n$ if p rotated around c by the angle $k2\pi/n$ lands in d_j (i.e., $\rho^k_c(p) \in d_j$). There is a perfect matching in G_r if and only if c is the center of a regular n-gon with vertices in the disks from D.

The above algorithm can be used for objects other than disks, only the running time will change depending on the complexity of the apex regions.

4.4.2 Optimization problem: Symmetry with imprecision

We now consider the following problem: Given a set $P = \{p_1, \ldots, p_n\}$ of n points, find minimum δ^* such that shifting each point by at most δ^* brings the points in symmetric position (which means they are vertices of a regular n-gon). We give an exact algorithm, a quick constant-factor approximation, and a PTAS for the problem.

(a) Exact solution

It is immediate that in the optimal solution, some J points of P are shifted by exactly δ^*; we argue that $J \leq 5$. Renumber the points in P so that the points shifted by δ^* are $p_1, \ldots, p_J$, and let $q_1, \ldots, q_J$ be the shifted points. Suppose

we know that q_j is the k_j-th vertex of the optimal n-gon, where $k_1, \ldots, k_J$ are some distinct integers between 1 and n. We can then write one equation for each $j = 1, \ldots, J$:

$$|R^{k_j \frac{2\pi}{n}}(p_j - c) - (q_1 - c)| = \delta^*$$

where c is the center of symmetry of the n-gon and $R^{k_j 2\pi/n}$ is the rotation matrix with rotation angle $k_j 2\pi/n$. Overall, we have J equations in 5 variables (two for each of c and q_1, and one for δ^*) . The system has a solution with an isolated δ^* when $J = 5$.

The above observations lead to a (high) polynomial-time algorithm for the problem: Guess 5 points of P and 5 numbers $k_1, \ldots, k_5$. For each guess, solve the above described system of 5 equations in 5 unknowns to get (a constant number of) candidate values for δ^*; for each candidate run the symmetry detection algorithm from Section 4.4.1 with radius-δ^* disks centered on points of P in the input.

(b) $O(1)$-approximations

We start with two auxiliary lemmas:

Lemma 4.4.1. *Let Q be an arbitrary regular n-gon; let g be its center. Let $r \in \mathcal{R}^{\in}$ be an arbitrary point; let q be the vertex of Q closest to r. Moving each vertex of Q by at most $|qr|$, the regular n-gon Q can be modified to a regular n-gon $Q_{g,r}$ that is also centered at g and has r as a vertex.*

Proof. Let the vertices of Q be $q, q_1, q_2, \ldots, q_{n-1}$ in counterclockwise direction. Consider the translation vector $\vec{pr}$ that moves p to r. Each vertex q_i of Q is translated by a vector that is defined by $\vec{pr}$ rotated by $i2\pi/n$ around g, see Fig. 4.15. Hence, each vertex is moved by a distance of $|qr|$ and the points $r, q'_1, \ldots, q'_{n-1}$ build a regular n-gon with center g, $Q_{g,r}$. □

Let $Q^* = q_1^*, \ldots, q_n^*$ be the optimal regular n-gon ($|p_i q_i^*| \leq \delta^*$), and let c^* be the center of Q^*. Let g be the centroid of P.

Lemma 4.4.2. $|c^* g| \leq \delta^*$.

Proof.

$$|c^* g| = \left|\left| \frac{1}{n} \sum_{i=1}^{n} q_i^* - \frac{1}{n} \sum_{i=1}^{n} p_i \right|\right| \leq \frac{1}{n} \sum_{i=1}^{n} ||q_i^* - p_i|| \leq \delta^*$$

□

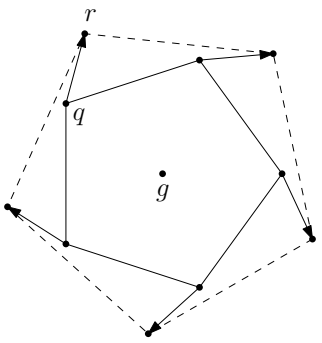


Figure 4.15: Q can be moved to $Q_{g,r}$ (the dashed polygon).

We are now ready to give our constant-factor approximation algorithms.

A 4-approximation Take any point $p \in P$ and compute, in $O(n)$ time, the regular n-gon $Q_{g,p}$ that has p as a vertex and g as center. Compute bottleneck matching between P and vertices of $Q_{q,p}$, i.e., find the ordering $q_1, \ldots, q_n$ of vertices of $Q_{q,p}$ and minimum $\delta_{g,p}$ such that for any $i = 1, \ldots, n, |p_i q_i| \leq \delta_{g,p}$.

Lemma 4.4.3. $\delta_{g,p} \leq 4\delta^*$

Proof. The n-gon $Q_{g,p}$ can be obtained from the optimal n-gon Q^* as follows: First, shift Q^* by $g - c^*$ (so that the center of the shifted polygon Q is at g), and then apply Lemma 4.4.1 (so that the polygon has p as a vertex). Let q^* be the vertex of the optimal n-gon Q^* closest to p. Before the shifting, we had $|q^*p| \leq \delta^*$. By Lemma 4.4.2, the shift is not larger than δ^*, and, hence, there is a vertex of the shifted polygon within distance $2\delta^*$ from p. By Lemma 4.4.1, $Q_{g,p}$ can be obtained from the shifted polygon, moving every vertex by at most $2\delta^*$. Overall, any vertex of $Q_{g,p}$ finds itself within distance $\delta^* + \delta^* + 2\delta^*$ from the corresponding point of P. □

Interestingly, constructing $Q_{g,p}$ alone does not yield a 4-approximation of the *value* of δ^* (even though we know that $Q_{g,p}$ is a 4-approximation); this is because (other than for p) we do not know which point of P moves to which vertex of $Q_{g,p}$. To know the value of $\delta_{g,p}$, one needs to compute the bottleneck matching between P and vertices of $Q_{g,p}$. While $Q_{g,p}$ itself can be computed in linear time, we know of no faster algorithm for computing $\delta_{g,p}$ than the general $O(n^{1.5} \log n)$-time algorithm of [53].

A 3-approximation To improve the approximation, run the above approximation algorithm with each point of P serving as the point p, and choose the one that leads to the smallest $\delta_{q,p}$ (overall, this algorithm takes $O(n^{2.5} \log n)$ time).

Lemma 4.4.4. $\min_{p \in P} \delta_{q,p} < 3\delta^*$

Proof. Consider the set of vectors $V = p_i\vec{q}_i^*, i = 1, \ldots, n$; they must "span the full 360^o" (formally, any vector in $\mathcal{R}^\in$ must be representable as a linear combination of vectors in V with non-negative coefficients). Thus, at least one vector $p^*\vec{q}^* \in V$ makes a positive angle with $c^*\vec{g}$ – the shift vector. Hence, the shift brings q^* closer to p^* – after the shift, the distance between the shifted vertex and p^* is smaller than it was before the shift, i.e., is smaller than δ^*. Applying the operations from Lemma 4.4.1 to the shifted polygon and p^*, moves each point of the shifted polygon by at most δ^*. Overall, any vertex of Q_{q,p^*} finds itself within distance $2\delta^* + \delta^*$ from the corresponding point of P. □

A PTAS Compute a 4-approximation δ of δ^*, and lay out $\frac{1}{\varepsilon} \times \frac{1}{\varepsilon}$ grids G_g and G_p in the δ-neighborhood of g and the δ-neighborhood of some point $p \in P$, respectively. Then, for each pair (g', p') of grid points from $G_g \times G_p$, compute the regular polygon $Q_{g',p'}$ centered at g' and having a vertex at p', and find the value $\delta_{g',p'}$ of the bottleneck matching between P and the vertices of $Q_{g',p'}$; this can be done in overall $O(\frac{1}{\varepsilon^4} n^{1.5} \log n)$ time.

Lemma 4.4.5. $\min_{g',p'} \delta_{g',p'} \leq (1 + O(\varepsilon))\delta^*$

Proof. Some vertex q^* of Q^* is within distance δ from p; thus, q^* is within distance $O(\varepsilon\delta^*)$ from some gridpoint $p^* \in G_p$. Shift the optimal polygon Q^* so that its center c^* moves onto the closest point $g^* \in G_g$. The shift moves each vertex of Q^* by $O(\varepsilon\delta^*)$; in particular, the shifted q^* remains $O(\varepsilon\delta^*)$-close to p^*. Applying Lemma 4.4.1, we obtain that each vertex of Q_{g^*,p^*} finds itself within distance $\delta^* + O(\varepsilon\delta^*) + O(\varepsilon\delta^*)$ from the corresponding vertex of P. □

4.5 Conclusion

We resolved a long-standing open question: Can one determine in polynomial time whether a set of objects has a convex transversal? We gave negative answers for segments and scaled copies of a convex polygon in 2D and for balls in 3D.

Our construction showing hardness of stabbing non-disjoint segments in 2D can be lifted to 3D while removing the intersections between the segments; hence, stabbing disjoint objects in 3D is also hard. Note that the segments/balls used in our hardness proofs are of drastically different sizes. But — at least for segments — our construction can be extended to the case where all segments have a length between 1 and $1+\epsilon$ for any $\epsilon > 0$. However, for unit line segments the construction fails. We leave this as an open problem.

On the positive side, we gave a polynomial-time algorithm to determine a convex stabber, if it exists, for a set of disjoint line segments or disjoint convex polygons under the restriction that the stabber vertices come from a given set C of candidate points. The most intriguing open question is whether the restriction can be removed.

In general, convex transversals open a whole new research direction. Apart from the algorithmic study, it could be of interest to investigate combinatorial properties of convex stabbers, e.g., the number of geometric permutations induced by convex stabbers for different classes of objects.

Chapter 5

Robust Trees and Highly Probable Path Problem[1]

Super-Dense Operations (SDO) is an operational concept for Next Generation Air Transportation System (NextGen) designed to enable very high arrival and departure throughput at metroplexes. In a metroplex, several airports are located in close proximity such that their collective arrival and departure operations form a coupled system. Given the increased throughput and complexity, it is critical to ensure that when off-nominal conditions occur, the safety of system can be maintained. (Emergency and failure conditions of the system are also part of the ongoing research, but not discussed in the chapter.) Off-nominal conditions are defined to be the situations in which all elements of the system are operating as designed, but operational or environmental factors are not as planned or forecasted. An example is the situation in which weather materializes differently than forecasted, thereby causing a significant reduction in arrival capacity.

In SDO-Airspace, arrival and departure air traffic flows follow tree-like routing structures that connect the outer boundary of the airspace with the metering fixes distributed along the boundary of terminal airspace, where metering fixes can be considered to be directly connected to the runway. (Between the fixes and the runway there are Terminal Radar Approach Control (TRACON)'s airspace and a final approach fix, which we will not discuss here.)

In this chapter, we study approaches to design and implement an enhanced tree-based routing structure that provides robustness to SDO. By robustness,

[1]This research is funded by NASA Ames Research Center under NRA contract NNA10DF52C, "Mitigation of Off-Nominal Events in Super Density Operations", and Metron Aviation.

we mean that our tree structure is better equipped for off-nominal conditions such as weather changes, in that it provides pre-computed operational flexibility properties to allow conduction of various mitigation strategies, such as route stretching, airborne holding, and comprehensive traffic flow deviation in extreme cases, such as airport closure. (We define mitigation strategies to be a collection of control actions necessary to adapt the SDO system to off-nominal conditions and then to return the system back to nominal conditions in a safe and efficient manner.) We describe the "Robust Tree Planner" prototype software that we developed to generate robust trees with respect to a spectrum of possible weather scenarios.

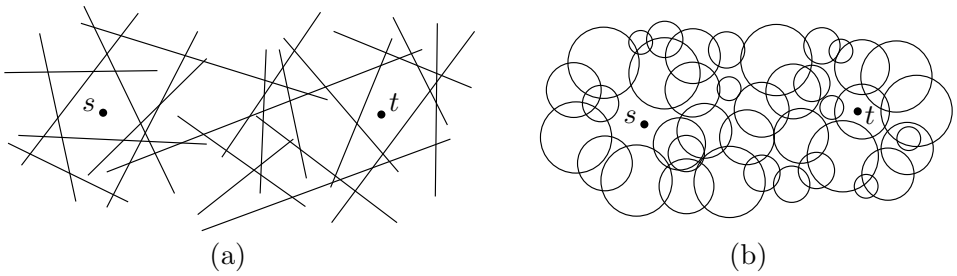
We also study the highly probable path problem, a theoretical problem that is directly abstracted from the application above. In robust tree routing, we compute a tree with respect to a set of weather instances, each of which is composed of a group of weather constraints and has an estimated probability of occurrence. The goal is to compute a tree whose edges and nodes have a high probability to be clear of constraints. The highly probable path problem deals with the specific case in which we are routing a single path. Then, in the case that the instances have the same occurrence probability, the question becomes: Find a path from start to goal that avoids as many sets of constraints as possible.

5.1 Problem Formulation and Results

The input to our HIGHLY-PROBABLE-PATH problem in the plane are a set of n geometric objects, S, and two points s (the source) and t (the sink) in the plane. We study the problem of finding a path from the source to the sink that intersects the minimum number of objects in S. In other words, our goal is to remove the minimum number of objects in S so that there exists an $s-t$ path that does not intersect any of the remaining objects.

We are particularly interested in the cases where the objects in S are line segments or (unit) circles. Fig. 5.1 shows 2 instances of the highly probable path problem. It is easy to see that in the first case, one needs to erase 4 segments to connect s and t. While in the second case, the answer is to remove 3 circles.

Our main results are as follows. We prove that the line segment case is NP-Hard and we present a fast and simple approximation algorithm for the unit disk case which can be extended to non-unit disk cases should the ratio between radii of the largest and the smallest disks is fixed.



(a) (b)

Figure 5.1: Definition of the highly probable path problem. (a) The case where objects in S are line segments. (b) The case where objects in S are circles.

5.1.1 Related Work

The segment version of the highly probable path problem is mainly discussed in the context of a series of optimization problems in segment arrangements. Independently to our work, Alt *et al.* [9] proved that the problem is NP-Hard and it is also NP-Hard to decide the minimum number of segments whose removal leaves the arrangement of the remaining segments with a single cell. They gave a polynomial time algorithm to solve the 2-CELL-SEPARATION problem. Kirkpatrick and Tseng [166] proved the NP-Hardness of the unit-length segment version. Kloder and Hutchingson [92] presented a polynomial time algorithm to find a minimum-sum-of-length set of segments in a polygonal domain that separates two regions.

The complexity of the disk version of the highly probable path problem remains unknown. However, it has been actively researched in the context of k-barrier path coverage problems in Wireless Sensor Networks [33] [37] [38]. Bereg and Kirkpatrick [21] studied the exact problem and give a 1.666-approximation under certain circumstances. Kumar *et al.* [113] proposed efficient algorithms to determine whether a belt region is k-barrier covered with a set of sensors and also considered coverage with high probability. In addition, Gibson *et al.* [66] studied a closely related problem of separating k disks and presented a O(1)-Approximation algorithm. The disk version of the highly probable path problem can be considered as path-coverage problem. For related problems of covering points, Brönnimann and Goodrich [28] and Mustafa and Ray [136] provided O(1) approximation and even PTAS.

5.1.2 Hardness Results for the Segment Case

Theorem 5.1.1. *Given two points s and t, and a set S of n arbitrary line segments in the plane, it is NP-hard to decide if there exists an $s-t$ path that intersects $k(k<n)$ of the segments.*

Proof. We prove the NP-hardness by reducing from INDEPENDENT SET. Let $G=(V,E)$ be an undirected graph with n vertices and e edges, the independent set (IS) problem asks if there exist k vertices in V such that no two of which are connected by an edge in E. We construct from graph G an instance of the highly probable path problem.

For each vertex of G, we construct a vertex gadget. As shown in Fig. 5.2(a), a vertex gadget has its top and bottom boundaries and a black rectangular area in between that creates two channels. For vertex v_i, we call the upper channel $C^u_{v_i}$ and the lower channel $C^l_{v_i}$. Note that all the boundaries and black areas in our proof are essentially dense segments that a path will never cross. Therefore, a path going from node S_{v_i} to node E_{v_i} in the vertex gadget must use one of the two channels. Finally, if the degree of v_i in G is $deg(v_i)$, we add $deg(v_i)$ segments, which will be connected to edge gadgets, that block the lower channel $C^l_{v_i}$, and $deg(v_i)-1$ short segments just blocking the upper channel $C^l_{v_i}$.

We then create a channel of n vertex gadgets aligned from left to right, forming the vertex part (Fig. 5.2(b)). The leftmost and rightmost nodes in the vertex part are S_{v_1} and E_{v_n}.

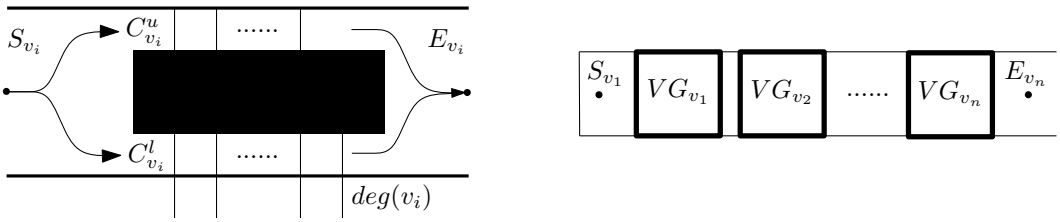


Figure 5.2: Vertex gadget construction. (a) The vertex gadget for vertex v_i. (b) Connecting the vertex gadgets together (VG_{v_i} is the vertex gadget for vertex v_i).

For each edge of G, we create an edge gadget similarly. As shown in Fig. 5.3(a), the edge gadget for $e_i=(v_i,v_j)$ also has top and bottom boundaries and a black area that separates the wide corridor from S_{e_i} to E_{e_i} into two

narrower channels, $C^u_{e_i}$ and $C^l_{e_i}$. Note that when approaching E_{e_i}, the original upper channel $C^u_{e_i}$ is below the original lower channel $C^l_{e_i}$. Two segments coming from the vertex gadgets for e_i's two endpoints v_i and v_j are used to block the two channels.

We then use segment-free channels having a similar shape to an edge gadget to connect the edge gadgets together, forming the edge part (Fig. 5.3(b)). The edge part starts at node S_{e_1} and ends at E_{e_m}.

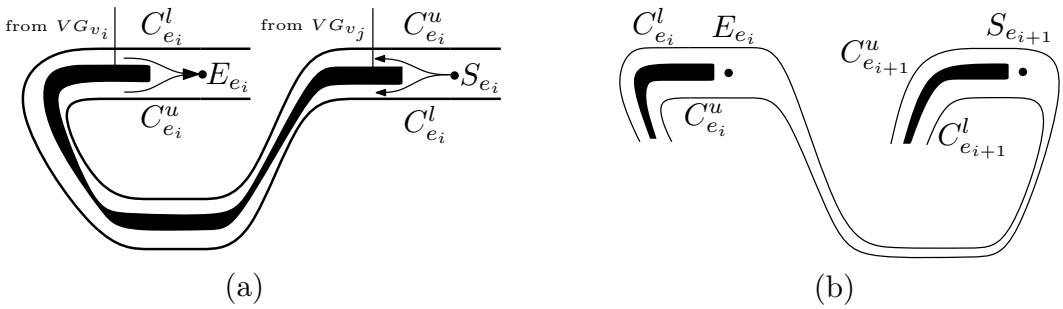


Figure 5.3: Edge gadget construction. (a) The edge gadget for edge $e_i = (v_i, v_j)$. (b) Using a segment-free channel from E_{e_i} to $S_{e_{i+1}}$ to connect the end of the edge gadget for e_i to the start of the gadget for e_{i+1}.

To finish the construction, we use a segment-free corridor from the end of the vertex gadget for v_n, E_{v_n}, to the start of the edge gadget for e_1, S_{e_1}, to connect the vertex part to the edge part. An example of the construction is shown in Fig. 5.4. In the constructed instance of highly probable path problem, the source s is S_{v_1} and the sink t is E_{e_m}. It is easy to see that an $s-t$ path must pass through all vertex and edge gadgets one by one.

We now prove that there exists an independent set of size k in G if and only if there is an $s-t$ path that intersects exactly $\sum_{v_i \in V} deg(v_i) - k$ segments in the construction. Or in other words, if and only if there is an intersection-free $s-t$ path after the removal of exactly $\sum_{v_i \in V} deg(v_i) - k$ segments in the construction.

First, suppose there is an independent set of size k in G. For each vertex in the independent set, we remove the segments blocking the upper channel in its vertex gadget. Hence the number of segments removed is $\sum_{v_i \in IS}(deg(v_i) - 1)$. Then in the remaining vertex gadgets corresponding to the vertices not in the independent set, we remove the segments that block the lower channels. The number of segments removed is $\sum_{v_i \notin IS} deg(v_i)$. Therefore, the total number of segments removed from vertex gadgets is $\sum_{v_i \in IS}(deg(v_i)-1)+\sum_{v_i \notin IS} deg(v_i) =$

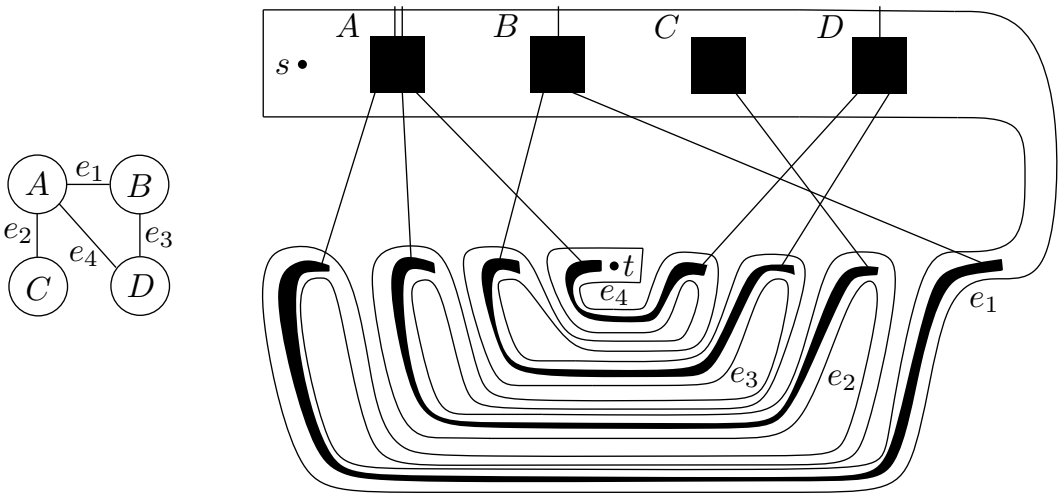


Figure 5.4: A complete example of the hardness gadgets construction from a graph.

$\sum_{v_i \in IS} deg(v_i) + \sum_{v_i \notin IS} deg(v_i) - \sum_{v_i \in IS} 1 = \sum_{v_i \in V} deg(v_i) - k$.

Because each edge has at most one vertex in the independent set, it has at least one vertex not in the independent set. Thus, at least one of the two channels in every edge gadget is segment-free. In other words, no segments need to be removed from edge gadgets. This proves that if there is an independent set of size k, after erasing $\sum_{v_i \in V} deg(v_i) - k$ segments in the constructed instance, there exists an $s - t$ path that does not intersect any segment.

On the other hand, suppose there is a segment-free $s - t$ path after the removal of $\sum_{v_i \in V} deg(v_i) - k$ segments. Without loss of generality, we assume that $\sum_{v_i \in V} deg(v_i) - l (n \geqslant l \geqslant k)$ segments are removed from the vertex part and $(l - k)$ segments are removed from the edge part. It follows directly from the construction that in the vertex part, the $s - t$ path uses l upper channels and $(n - l)$ lower channels to go from S_{v_1} to E_{v_n}. In edge part, let $e_s = (v_{ss}, v_{se})$ be an edge both channels of whose gadget are blocked by segments. Then the $s - t$ path must clear one of the channels to get through. Without loss of generality, suppose it is the segment from vertex gadget $VG_{v_{se}}$ blocking the upper channel $C^u_{e_s}$ that is erased, which implies that the $s - t$ path must have used the upper channel in $VG_{v_{se}}$. But what if the path is modified to use the lower channel $C^l_{v_{se}}$? In that case, the upper channel $C^u_{e_s}$ in edge gadget for e_s is free. The segment originally removed from $C^u_{e_s}$ is now removed from $C^l_{v_{se}}$ in $VG_{v_{se}}$ instead. It is easy to see that the total number of segments erased remains unchanged, but the $s - t$ path now uses one more, $(n - l + 1)$, lower

channels in the vertex part and removes one less segments in the edge part. (Note that the total number of removals can be fewer in case that more than one segments originally removed from the edge part come from the same vertex gadget.)

By repeating the process, all the segment removal operations that were done in the edge part are now conducted in the vertex part, while the total number of removals is at most $\sum_{v_i \in V} deg(v_i) - k$. Consequently, the $s-t$ path goes through at least k upper channels in the vertex part. And those vertex gadgets whose upper channels are used correspond to an independent set of size at least k. Hence there exists an independent set of size k in G. □

Because the independent set problem is hard to approximate within a logarithmic, this proof also suggests that the highly probable path problem for segments is hard to approximate.

5.1.3 Exploring the Disk Case

In the highly probable path problem for disks, we are interested in finding an $s-t$ path intersecting the minimum number of disks (Fig. 5.1(b)). As part of future research, our conjecture is

Conjecture 5.1.2. *Given two points s and t, and a set S of n arbitrary disks in the plane, it is NP-hard to compute the minimum number, k ($k < n$), of disks whose removal connects s and t with a path intersecting none of the remaining $(n-k)$ disks.*

Bereg *et al.* [21] proves the following lemma: For an arbitrary disk d_i in S, a highly probable $s-t$ path crosses the boundary of d_i at most four times, given (1) either s or t is at least $\sqrt{3}-1$ away from the boundary of d_i; (2) neither s nor t is in d_i. In other words, the $s-t$ path enters and exits the boundary of d_i at most twice. Fig. 5.5(a) shows an example where the boundary of a disk is crossed four times and the minimum number of disks one needs to remove is one.

The lemma leads to a straightforward 2-approximation algorithm. Given the arrangement of disks, we build up its directed weighted dual graph $\bar{G}$ which has a vertex corresponding to each face in the arrangement and an edge joining two vertices whose corresponding faces are neighbors (a pair of faces that can be connected by removing one disk). Fig. 5.5(c) shows the dual graph (undirected version) of the arrangement in (b).

In $\bar{G}$, edges that go into another disk are assigned weights 1, while all the other edges have weights 0. (By an edge $(f_i \to f_j)$ going into a disk d_r, we mean that $(f_i \to f_j)$ crosses the boundary of d_r and $f_i \notin d_r$, $f_j \in d_r$.) As shown in Fig. 5.5(d), the solid directed edges have weights 1, while the dashes ones have weights 0.

We now compute the shortest path from vertex f_s to vertex f_t in $\bar{G}$, where f_s's and f_t's corresponding faces in the arrangement contain s and t, respectively. Since each disk's boundary can be entered and exited at most twice, the length of the shortest path is at most $2k$, hence a 2-approximation.

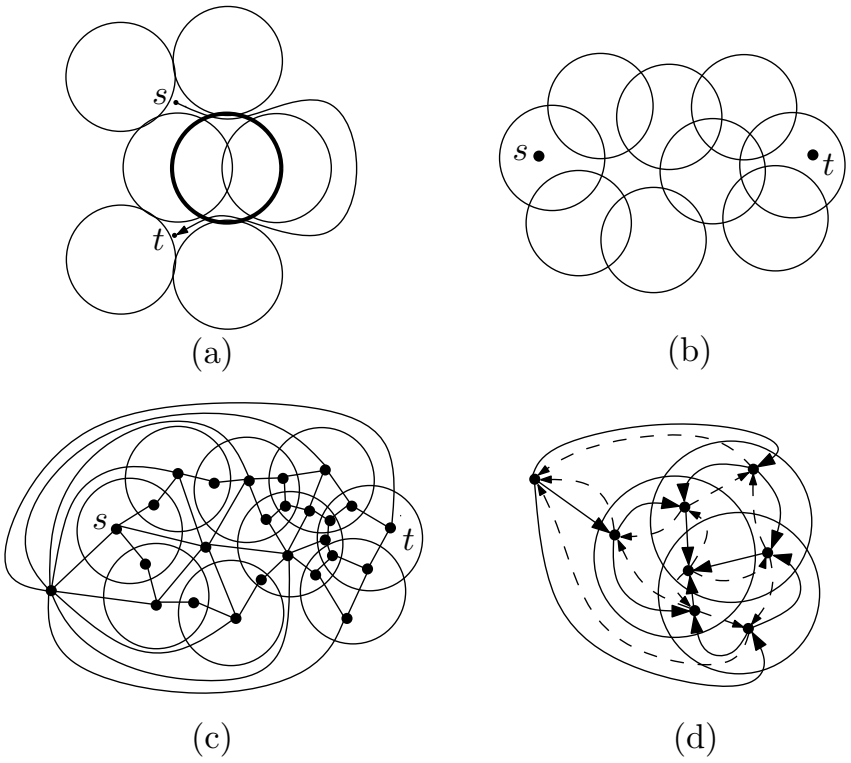


Figure 5.5: Dual graph construction. (a) An $s - t$ path which intersects only one disk (thick boundary). (b) An arrangement of disks. (c) The directed dual graph of the arrangment in (b). (d) Assigning weights to the edges in the directed graph.

5.2 Practical Heuristics: Implementation

We have implemented algorithms for robust tree generating and applied them to weather data for use in super dense operation(SDO) experiments and in NASA's FACET simulation tool. The FACET-based scenarios are performed in collaboration with colleagues at Metron Aviation. The implementation is in C++ and the the heuristic is based on bottommost filling of SDO airspace with tree branches. We call the prototype software "The Robust Tree Planner".

The Robust Tree Planner implements an algorithm for computing "robust" arrival trees within SDO airspace populated with weather cells. We describe each component of the algorithm below, including the grid generation, the search graph generation, the Phase 1 tree computation, the Phase 2 tree optimization, and operational flexibility properties computation.

The SDO airspace is mathematically modeled as an inverted cone shape (Fig. 5.6(a)). While this cone indicates a constant descent rate, which is unrealistic, our model generalizes directly to the case of a prescribed descent profile specifying the altitude of arriving flights, as a function of the distance from the airport. (Part of the future research is to design a more complex model to address the situation in which different flights can have substantially varying descent profiles.) The apex C sitting on the ground represents the airport runway and the periphery of its inverted base represents the outer boundary (entry ring) of the airspace, the altitude of which is $|CC_o|$. The inner boundary of the airspace, where the metering fixes are, is a range ring in between the base and the apex, parallel to the base. We define a quadrant to be a quarter (90-degree section) of the surface of a cone. The Robust Tree Planner computes trees within quadrants, in between the outer boundary $\overrightarrow{A_oB_o}$ and the inner boundary $\overrightarrow{A_iB_i}$ (Fig. 5.6(b)).

The weather constraints come from forecasts that are grid-based, indicating for each pixel at a certain altitude what the intensity value is. From the forecast data, using the CWAM model, we obtain a grid of cells in 3D with associated probabilities of avoidance. Thus, in our mathematical model, we consider a very simple model in which weather constraints are modeled as uncertain rectangular axis-aligned boxes with square top and bottom faces, at the altitude extremes of the weather cell. (We are using a Cartesian coordinate frame, not accounting for the curvature of the earth.) Note that weather cells can, in general, overlap in our model; if they arise from CWAM data, though, the cells are pairwise-disjoint. For experimentation purposes, we allow weather cells that are more general, and our software includes the option to generate

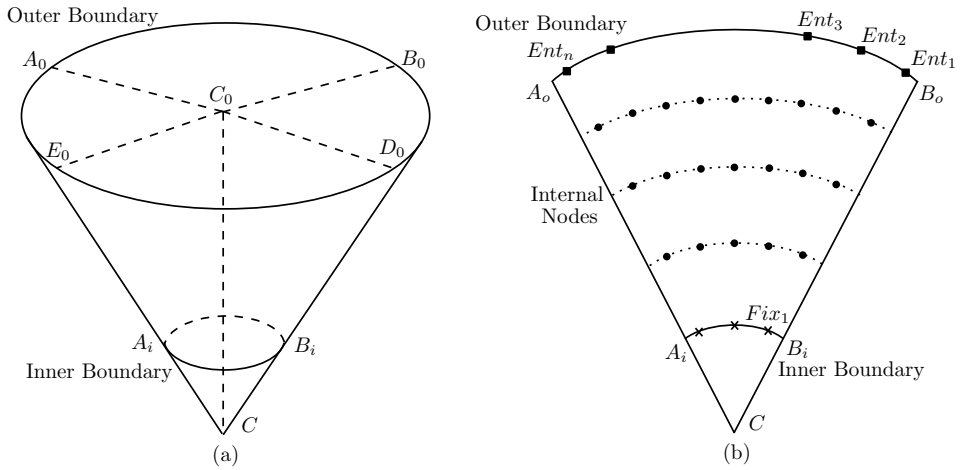


Figure 5.6: The mathematical model of SDO Airspace and quadrants. (a) An inverted cone shape is used to represent the SDO airspace. The base of the cone centered at C_o is the outer boundary. The inner boundary (range ring) is the circle centered at C_i parallel to the base. (b) A quadrant A_oB_oC is shown, with entry nodes (indicated with small, solid squares), fix nodes (small crosses along $\overrightarrow{A_iB_i}$) and internal nodes (small solid circles). The quadrant is a quarter of the cone shape shown in (a). (For illustration only; the figure is not to scale and distance measurements are not precise.)

random sets of weather cells. Therefore, A cell can be specified by the position of the lower left vertex of its bottom face B_1 and its width cw and height ch (Fig. 5.7(a)). In addition, each cell has an associated deviation probability value, dev_p, indicating the estimated probability that flights will avoid the cell as given by the CWAM model. In general, the more severe the weather is in the cell, the more likely a flight is to avoid it. A weather instance is given by a set of weather cells (Fig. 5.7(b)). A weather ensemble E is given by a group of N weather instances W_1, W_2, ..., W_N, each with an associated probability value p_{wi} estimating the likelihood that the instance is realized. (Naturally, the probabilities sum to one: $\sum_{i=1}^{N} p_{wi} = 1$).

5.2.1 Grid Generation

The Robust Tree Planner algorithm utilizes a discretization of airspace in its search and optimization. This discretization is based upon establishing discrete grid points of three types: entry nodes, arrival fix nodes, and internal merge

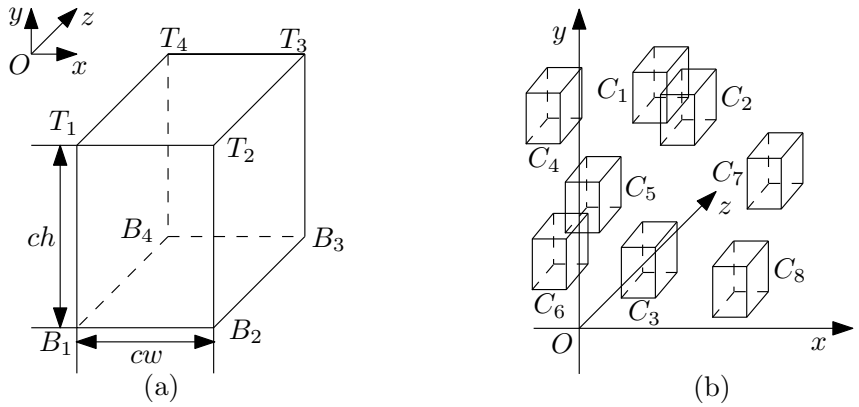


Figure 5.7: Weather constraints representation. (a) A weather cell of width cw, height ch. (b) A weather instance consisting of 8 weather cells C_1, C_2, ..., C_8.

nodes. All grid points are on the surface of the cone (quadrant), as shown in Fig. 5.6.

Entry Nodes

A user-specified parameter n determines the number of equally spaced arrival nodes (entry segments) on the outer boundary of the SDO airspace. We index the nodes counter-clockwise using Ent_1, Ent_2, ..., Ent_n. Associated with each entry node Ent_i is an RNP (required navigation performance, representing the width of the tree branch) requirement, RNP_i, indicating the maximum RNP for arriving flights to that entry node over the look-ahead planning horizon. Our search algorithm establishes flight legs consisting of air lanes of clearance width $2 \times RNP_i$ (Fig. 5.8(a)) for the path in the tree from entry node Ent_i, through the tree, to the arrival fix. The first (resp., last) entry node Ent_1 (resp., Ent_n) has distance $2 \times RNP_1$ (resp., $2 \times RNP_n$) to the right (resp., left) boundary of the quadrant, along the outer range ring (Fig. 5.8(a)). All other entry nodes are equally spaced along the outer range ring of the quadrant, or picked from a set of equally spaced candidate positions along the outer range ring. All entry nodes have altitudes the same as that of the outer boundary of the quadrant ($|CC_0|$ in Fig. 5.6(a)). If there is insufficient room to space the nodes while respecting the RNP values, the software reports "The quadrant cannot accommodate the RNP requirement."

Arrival Fix Nodes

The Robust Tree Planner allows up to three arrival fix nodes. We require the fix nodes to accommodate at least the maximum RNP requirement, RNP_{max}, among the associated entry nodes. We search for feasible placements of the fix nodes, starting from the center point of the inner boundary of the quadrant. (In this step, "feasible" placements are the positions centered at which the RNP_{max}-radius balls do not overlap with weather constrained cells, as shown in Fig. 5.8.) We continue the search by walking to the left and right simultaneously. We search for at most three fix nodes. If there is no fix node that satisfies the RNP requirement, the tree planner reports "A tree cannot be generated." We have the option to require that arrival fix nodes are spaced at least a certain minimum distance from each other.

Internal Merge Nodes

Our goal is to generate enough internal nodes to serve as a rich set of candidate merge nodes, while not overcrowding the airspace, causing the search graph to explode in size. We use the minimum RNP requirements RNP_{min} among entry nodes as a key parameter. The search graph has multiple layers, organized in concentric circular arcs on the surface of the cone (quadrant). Adjacent layers have separating distance at least $3 \times RNP_{min}$ (Fig. 5.8(c)). Based on this requirement, we generated as many layers as possible within the SDO airspace. We let r_q denote the radial thickness of the quadrant; i.e., r_q equals the difference between the distance to the apex of the cone from the outer range ring and that from the inner range ring ($|A_oA_i|$ or $|B_oB_i|$ in Fig. 5.6(b)). Then, the number of layers in our search graph is equal to the maximum integer n_l that satisfies $(3 \times RNP_{min}) \times n_l < r_q$. Next, we generate internal nodes on each of the n_l levels. A pair of adjacent internal nodes is separated by at least $2 \times RNP_{min} + 2$. Thus, if the arc length of the current layer is arc_i (which can be computed easily), the number n_{li} of nodes on a specific layer is the maximum integer that fulfills $(2 \times RNP_{min} + 2) \times n_{li} < arc_i$ (Fig. 5.8(c)). To generate the nodes in each layer, we partition the arc equally into $2 \times n_{li}$ parts, resulting in $(2 \times n_{li} + 1)$ candidate positions, and place the n_{li} nodes at the 2nd, 4th, 6th, ..., $2 \times n_{li}$th positions (the first and last candidate positions are on the right and left boundaries of the quadrant, respectively).

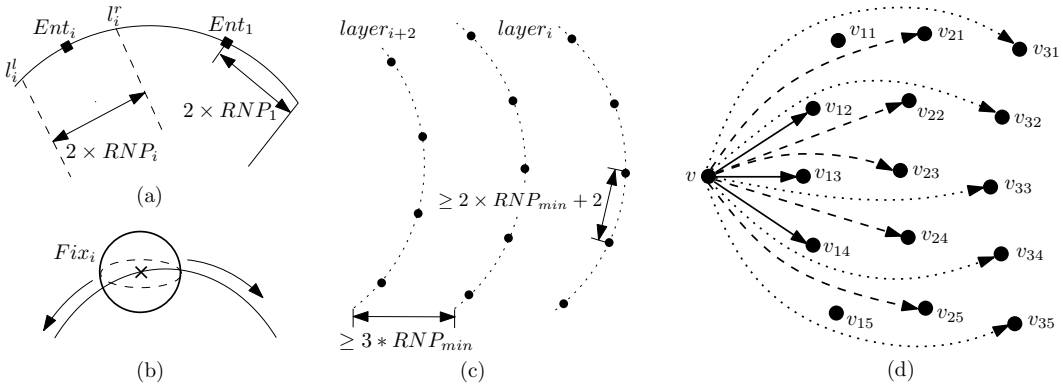


Figure 5.8: Search graph generation. (a) Generating entry nodes on the outer boundary of the quadrant. (b) Generating fix nodes on the inner boundary. (c) Generating layers between the outer and inner range rings and generating internal merge nodes on each layer. (d) Generating graph edges that connect pairs of grid nodes. Node v has outgoing edges to 3 nodes in the next level, 5 nodes each to the 2nd and 3rd next levels, 13 in total.

5.2.2 Search Graph Generation and Test Graph against Weather Constraints

After determining the grid nodes as described above, we generate a directed acyclic graph (DAG) as our search graph by making selected edge connections among the nodes. The rule is as follows: a node at level i has outgoing edges to selected nodes at the three succeeding levels, $(i + 1)$, $(i + 2)$, and $(i + 3)$. (The parameter "3" can be modified.) For a specific node N_i at level i, we find the closest node to N_i at each of the next 3 levels, say nodes N_{i+1}, N_{i+2} and N_{i+3}. Then N_i is designed to have outgoing edges to N_{i+1} and its left and right neighbors. Node N_i also has outgoing edges to N_{i+2}, N_{i+3} and their two direct left and two direct right neighbors. Therefore, a typical node has 3+5+5=13 outgoing edges (Fig. 5.8(d)). If any of the 13 nodes do not exist, no action will be conducted to search for a replacement. Instead, we simply ignore them. The rule is slightly different for incoming edges to fix nodes: Only the nodes on the last two levels have outgoing edges to fix nodes, and they are connected to each of the fix nodes.

Given a weather forecast ensemble E, together with an RNP value rnp and a deviation threshold value dev_{thres}, we test each edge in the graph against each member of the ensemble to compute the probability that a cylinder corresponding to a flight leg of radius rnp, centered on the edge segment is

clear of weather constraints; similarly, we test each node in the graph against each member of the ensemble, computing the probability that a ball of radius rnp, centered at the node, is clear of weather constraints. Suppose E is composed of a set of weather instances W_1, ..., W_N, each of which has an associated probability p_{wi}, with each instance W_i consisting of a set of weather cells C_1,..., C_m, each having an associated deviation probability dev_{pi}.

Our current implementation of the algorithms to test weather cell avoidance are based on two-dimensional computations, in the horizontal plane, based on projecting search graph edges and weather cells into the plane. (Since our weather data will be given to us at certain fixed altitude layers, our three-dimensional computation in fact is partitioned into a set of two-dimensional computations between horizontal slices of weather data; thus, our two-dimensional implementation is almost fully general in three dimensions. Plus, 3D collision detection algorithms between cylinders or balls and weather cells are unacceptably slow and unnecessarily complicated.)

We test an edge $e_i = (v_i, v_j)$, with RNP requirement RNP_{ij}, against a weather ensemble E by testing the edge against each weather instance in E. For a specific weather instance W_I, we use a variable res_i to denote the test result of whether e_i is free ($res_i = 1$) of weather constraints in W_I or not ($res_i = 0$). After conducting the test against all of the W_I's, the overall probability of clearance is obtained by conditioning on the instance, yielding $p_{ei} = \sum_{i=1}^{n}(p_{wi} \times res_i)$.

In order to test if e_i is in conflict with a weather cell C, we first compare the deviation threshold dev_{thres} with C's deviation probability, dev_p. The test is continued only if $dev_{thres} < dev_p$, since, otherwise, the weather cell is not severe enough and the result is "no conflict". Next, as a heuristic to accelerate conflict detection, we compare the axis-aligned bounding box of the edge with the (already axis-aligned) weather cell box; we proceed with further computation only if the boxes overlap. (More sophisticated methods of intersection testing, e.g., using bounding volume hierarchies or range search data structures are possible and may be investigated in future research.)

Given a cell C, we first extract the relevant subsegment, $e'_i = (v_{i1}, v_{j1})$, of $e_i = (v_i, v_j)$ that lies in between the top and bottom (horizontal) faces, S_u and S_l, of C; see Fig. 5.9(a). (In case e_i only intersects one of the faces S_u and S_l, depending on which one it intersects, e_i could be (v_{i1}, v_j) or (v_i, v_{j1}).) Next, we project e_i on S_l and call the projected subsegment (v'_{i1}, v_{j1}). Then we determine if the rectangle of width $2 \times RNP_{ij}$ centered at (v'_{i1}, v_{j1}) intersects with the bottom, B_C, of C (Fig. 5.9(c)).

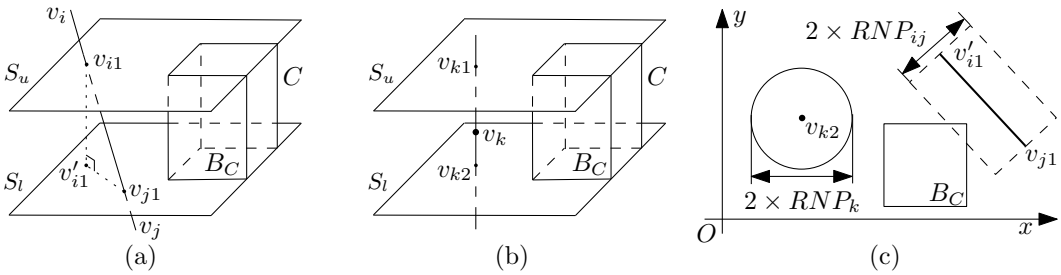


Figure 5.9: Testing edges/nodes against weather cells. (a) Weather clearance testing between an edge and a weather cell. (b) Weather clearance testing between a node and a weather cell. (c) Clearance testing after projecting nodes and edges on the plane S_l.

A similar sequence of tests is done for nodes of the search graph: For a node v_k with RNP requirement RNP_k (which models the required clearance at this location, if it is to be used as a merge node), we consider the node to have a conflict with a weather cell if v_k lies vertically in between S_u and S_l (i.e. the altitude of the node places it within the range of altitudes spanned by the weather cell) and the (two-dimensional) disk of radius RNP_k centered at the projection node v_{k2} intersects the projection, B_c, of the weather cell (Fig. 5.9(b)(c)). Then, the probability of clearance associated with v_k is $p_{v_k} = \sum_{i=1}^{n}(p_{wi} \times res_i)$.

Each edge and node of the DAG stores this probability. Furthermore, for each edge of the DAG, we compute the probabilities of having operational flexibility airspace of a width w to the left and to the right (separately) of the centerline (detail in section 5.2.5).

5.2.3 Phase 1 Tree Computation

Phase 1 of our algorithm is based on a tailored depth first search (DFS) in the search graph, utilizing only those edges and nodes of the DAG that satisfy a user-specified robustness criterion, which is represented by a threshold probability on the availability of a lane with the RNP width, $p_{nodeEdge}$. Consequently, a generated tree must have all its nodes (resp., edges) to satisfy $p_{v_k} \geqslant p_{nodeEdge}$ (resp., $p_{e_i} \geqslant p_{nodeEdge}$), where p_{v_k} (resp., p_{e_i}) is the probability of clearance. Other examples of such criteria include a threshold probability on the probability of availability of sufficient operational flexibility airspace on either side of the flight leg.

The search starts from each entry node from Ent_1 to Ent_n, giving priority

to the edges that go towards the right side of the quadrant (viewing from inner to outer range ring, B_0B_i in Fig. 5.6(b)), as shown in Fig. 5.10(a). We also give secondary priority to long edges because the search graph is fine enough that we want the flight segments of tree branches to be reasonably long, both for compactness of the tree description and for having sufficient length between consecutive merge points. We stop the DFS as soon as we reach an arrival fix node. After completing DFS starting from Ent_i, we start a new DFS from Ent_{i+1}.

There is a condition that can trigger the early termination of a DFS: when a new branch goes to a node tr_{ij} that is on the previous branch (by "previous", we mean if the current branch starts from Ent_i, the one starting from Ent_{i-1}). As long as (1) node tr_{ij} has in-degree less than 2 (usually at most 2 air lanes are allowed to merge together, but the parameter can be adjusted), and (2) the remaining part of the previous branch can accommodate the new RNP requirement, we stop the DFS immediately and the new branch merges into the previous branch.

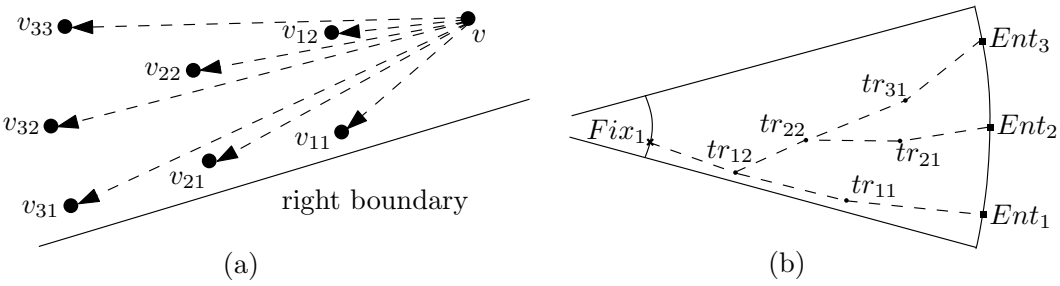


Figure 5.10: Bottommost tree computation. (a) When the graph search algorithm looks for the next node from v, the priorities from high to low of the outgoing nodes are: v_{11}, v_{21}, v_{31}, v_{32}, v_{22}, v_{12}, v_{33}. (b) The problem with phase 1 bottommost trees: branches tend to bias towards the right boundary of the quadrant, hence highly suboptimal.

When a DFS cannot continue (i.e., the stack is empty and the search has not reached an arrival fix node), the software reports that a tree cannot be found that accommodates the demand input with the specified RNP requirements. In future work, we will add a dynamic programming component to the algorithm to optimize over the possibilities when the demand cannot be fully satisfied; the full model involves a complex, multicriteria optimization problem.

5.2.4 Phase 2 Tree Optimization

The Phase 1 tree generation determines the availability of a robust routing tree; however, the tree is not optimized for length and appears to be highly suboptimal (Fig. 5.10(b)). Our Phase 2 algorithm optimizes the merge tree using a tautening function to "pull taut" (locally shorten) the branches of the Phase 1 tree. We tauten the branches in the opposite order of bottommost branches, starting from Ent_n to Ent_1. An important property that should be satisfied is that as long as there is a bottommost tree, this tautening algorithm should always produce a better tree; it cannot fail.

When tautening the ith branch, we conduct a shortest path algorithm starting from the ith entry node Ent_i. We use the previous branch (or the right boundary) and the next branch (or the left boundary, with definition of next branch similar to that of previous branch) as "soft" constraints, which means we cannot cross them, but we are allowed to use the nodes that are part of them to enable natural merging (Fig. 5.11). Again, the merge nodes must have in-degree at most 2, and that once the current branch merges into the next branch, it cannot leave that branch (by the property of a tree). Therefore, when a possible merging happens, we test the remaining part of the next branch to see if it can accommodate the new requirement of RNP_i. If not, the potential merge node cannot be used. The algorithm finds the shortest feasible path in between these "soft" constraints that connects Ent_i to a fix node.

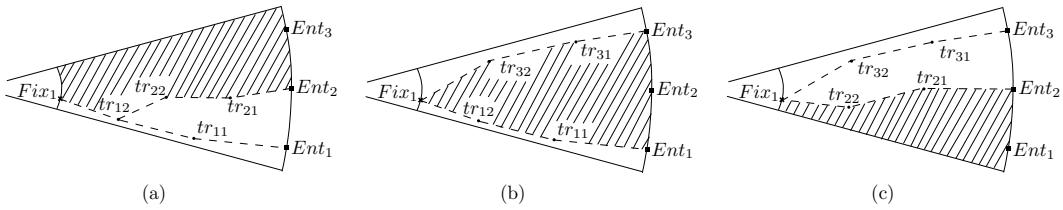


Figure 5.11: Three steps to optimize the bottommost tree in Fig. 5.10(b). The shadowed regions are where the shortest path algorithm operates. (a) "Soft" constraints are the left quadrant boundary and the bottommost branch starting from Ent_2. (b) "Soft" constraints are the tautened branch starting from Ent_3 and the bottommost branch starting from Ent_1. (c) "Soft" constraints are tautened branch starting from Ent_2 and the right quadrant boundary.

The algorithm above works well in most cases in which weather is not too severe. However, there is an undesirable behavior: the branches generated tend to merge late (near the arrival fix nodes). For the case in which there are a lot

of weather cells near the fix nodes, the region near the arrival fixes may be a bottleneck through which there is only room for one branch to go. Then, it is very likely that the Phase 1 tree branches must merge very early in order to form a single branch well before reaching the bottleneck area, hence the arrival fix nodes. However, the Phase 2 algorithm, which employs a shortest path algorithm for tautening branches, is a form of "greedy" algorithm, attempting to route tree branches to a fix node as soon as possible. Hence, the first few tautened branches may have already "filled up" the bottleneck area, causing all the tree nodes near the bottleneck area to have in-degree 2, so that the subsequent branches have no possibility to merge into the taut branches of the tree before the bottleneck. This problem causes the Phase 2 algorithm to fail. To solve this problem, we employ a simple new method to relax the "greediness" of the algorithm.

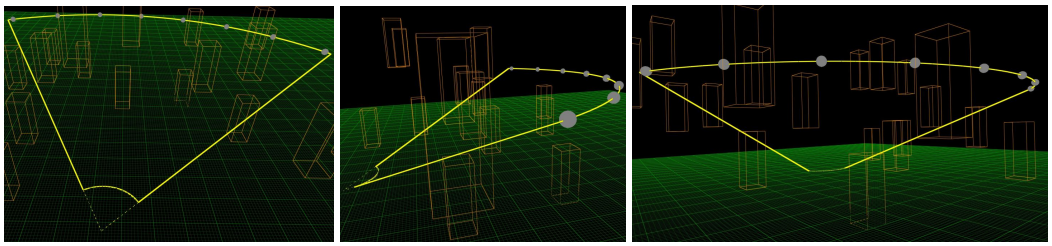
We use a new parameter called $mergeEarly$: when $mergeEarly$ is j, for a specific branch i that is tautened by the shortest path algorithm, the $(i+1)$th, $(i+2)$th, ..., $(i+j-1)$th branches will try to merge into the ith branch as early as possible. For example, when $mergeEarly$ is 2, then every other branch attempts to merge to its next branch as early as possible.

The technique used in early merging is simple: As soon as the shortest path algorithm reaches a node v_m belonging to the next branch, and the part of the next branch from v_m to its fix node is able to accommodate the new RNP requirement, the algorithm stops and the current branch merges into the next branch. It is easy to see that the new algorithm is effective because now a branch always merges into the next branch at least as early as the original branch generated by the tautening algorithm without $mergeEarly$. And in most cases, much earlier.

We start the tautening process with $mergeEarly$ set to 1, meaning that the original greedy approach is tried first. If the algorithm fails to optimize the tree, $mergeEarly$ is increased by 1. We continue in this way until $mergeEarly$ is $(n-1)$. Through experimentation, we have found this revised algorithm to be quite effective, always generating an optimized tree for a given Phase 1 tree.

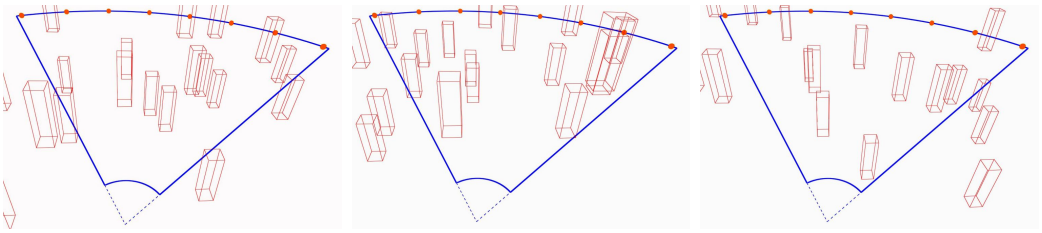
5.2.5 Operational Flexibility

After the generation of the Phase 2 optimized tree, a set of j user-specified width values w_1, w_2, ..., w_j are used to generate probability distributions quantifying operational flexibility properties for each edge of the tree: the operational flexibility wiggle airspace for holding patterns (left of the edge), the operational flexibility airspace for path stretch maneuvers (right of the edge), and the



(a) (b) (c)

Figure 5.12: Viewing a quadrant from different angles. The green plane is $z = 0$. (We use black background color to emphasize the 3D effect.) The grey balls along the outer boundary mark the entry node positions and their corresponding RNP requirements. (a). Front View. (b). Side View. (c). Back View.



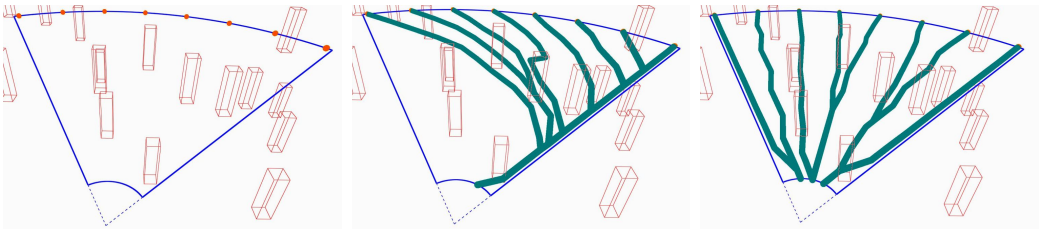
(a) (b) (c)

Figure 5.13: Example of a weather ensemble consisting of 3 instances, shown in (a)(b) and (c).

operational flexibility airspace centered along the edge (extending on both sides of the edge). For each property, we compute j probabilities of clearance, one for each choice of w_i. The way to compute the probabilities is the same as testing edges and nodes against weather ensembles, described in section 5.2.2 and Fig. 5.9.

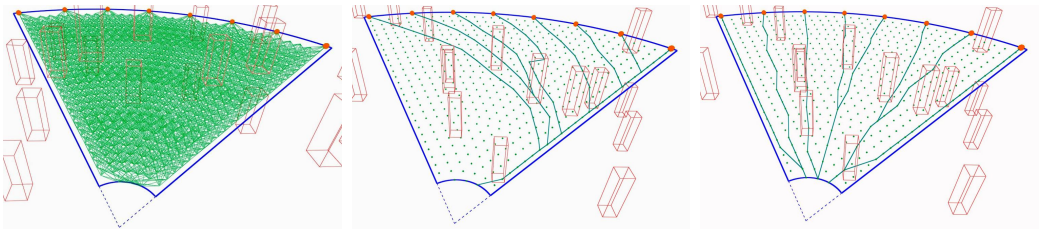
For each edge, we compute and store j probability values corresponding to operational flexibility airspace for holding patterns. In other words, if w_{HP} is a random variable specifying the width of a maximum-width constraint-free rectangle to the left of an edge, we are computing the probabilities $p(w_{HP} > w_i)$, for each w_i, yielding an approximation of the cumulative distribution function of the random variable w_{HP}. Similar computations are done for rectangles of width wi to the right of and centered on each edge.

Additionally, off-nominal exit points (ONEPs) are generated along edges of the tree, spaced evenly according to a spacing parameter that depends on



(a) (b) (c)

Figure 5.14: Examples of Phase 1 and Phase 2 trees. Orange balls along the outer boundary of the quadrant represent entry nodes. Their radii correspond to the RNP requirements (a) The front view of the quadrant. (b) The Phase 1 bottommost tree. (c) The Phase 2 optimized tree.

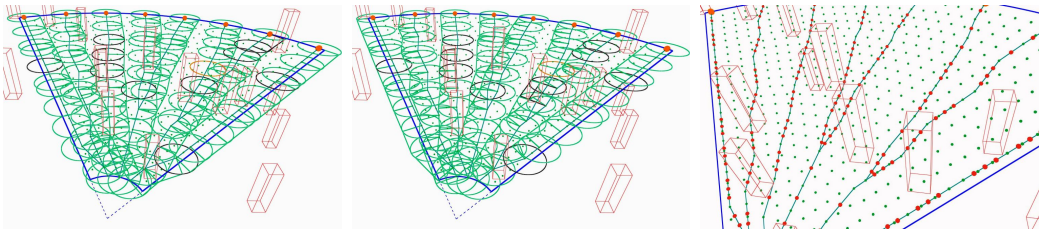


(a) (b) (c)

Figure 5.15: The underlying routing DAG (directed acyclic graph) used to generate the trees shown in Fig. 5.14. (a) The DAG (shown in green). (b)(c) The center lines of Phase 1 and Phase 2 tree branches, shown together with layers of nodes in the DAG.

the length of each edge and its associated RNP value *rnp*. The current default is that each edge has at most 3 evenly spaced ONEPs along the edge, with the requirement that the radius-rnp disks centered at the ONEPs are disjoint, so that ONEPs are not placed unnecessarily close to each other. The number of ONEPs could be 0, 1, 2, or 3, depending how the edge length compares with *rnp*. The ONEPs are prepared for the next phase of experiments, in which contingency trees to alternate airports are generated. In those trees, the ONEPs are used as the new entry nodes.

This computed information is stored with each tree edge so that our algorithm can reason about the probabilities of availability of different sizes and types of operational flexibility airspace for holding patterns and path stretch maneuvers, and can compute contingency trees that allow diversions to alternate fixes or airports.



(a) (b) (c)

Figure 5.16: Examples of operational flexibility properties. (a) Wiggle airspace for holding patterns (left of the center lines). (b) Path Stretching airspace (right of the center lines). (c) Deviation nodes on the center lines, shown in small red dots.

The same set of operations are conducted on each node of the tree as well. We compute for a set of disks centered at the node with radii w_1, w_2, ..., w_j, the probability of clearance. The information is stored with each tree node as its operational flexibility property.

5.2.6 Illustrations

Fig. 5.12– 5.16 illustrate images from the Robust Tree Planner software. Fig. 5.12 shows a 3D quadrant viewed from different angles and Fig. 5.13 shows a randomly generated weather ensemble consisting of 3 instances. The underlying search DAG and examples of generated Phase 1 and Phase 2 trees are shown in Fig. 5.14 and Fig. 5.15.

Fig. 5.16 illustrates 3 types of operational flexibility properties. Different colors illustrate the free-of-constraints probabilities of the corresponding rectangles (for edges) or disks (for nodes); green indicates high probability (at or near 1) that the area is free of weather, dark grey indicates lower probability, orange indicates lower still, and dark red indicates lowest non-zero probability.

5.3 Practical Heuristics: Experiments

We report experimental results involving algorithmic design choices. The user interface allows one to use a mouse to adjust the quadrant, setting its center position, inner and outer radii, inner and outer heights, and starting angle relative to the positive x axis. The weather data used for testing can be real weather from NASA's FACET simulation tool, or can be randomly generated

by our random weather generator. The key parameters specifying a weather ensemble are: (1) The number of weather instances $numInstances$; (2) The probability that each instance happens p_{wi}, which sum up to 1. (3) The number of weather cells in each weather instance $numCells$; (4) Range of cell widths $cellWidth$; (5) Range of cell heights $cellHeight$; (6) The deviation threshold of each weather cell dev_{thres}, and (7) a bounding box of all the cells specifying approximately where they are. Our random weather generator simulates each of these parameters based on the position of the quadrant and a set of user input ranges.

5.3.1 Speed Tests

In our tree generating speed tests, the quadrant has inner radius 15, outer radius 100, inner height 0 and outer height 37. (The position, size and height of the quadrant is not important in the experiments because the parameters of weather ensembles can always be adjusted accordingly. The reason we choose to set the quadrant with the group of parameters above is that they allow us to use the user interface to adjust and view the quadrant most efficiently.) The parameters used to generate simulated weather ensembles are carefully chosen so that the instances have their cells evenly distributed, neither overly crowded nor overly light. Ensembles are generated within the axis-aligned bounding box of the quadrant, with $numInstances$ between 1 and 5, the same $p_{wi} = 1/numInstances$ across all the instances, $numCells$ between 5 and 17, $cellWidth$ between 5 and 15 and $cellHeight$ between 10 and 20, all uniformly distributed. The deviation probabilities (dev_p) of cells are defaulted to 1 so that every cell always has its impact on the tree generating.

The two important parameters for trees are rnp, measuring the thickness of a tree, and $p_{nodeEdge}$, the threshold probability criterion no less than which each node and edge in the tree must have its probability of clearance be. rnp (resp., $p_{nodeEdge}$) is randomly generated to be uniformly distributed between 1 and 4 (resp., 0.6 and 1.0).

We tested 2000 randomly generated test cases on a 2.26GHz Intel CPU computer. Trees are successfully generated in 1505 cases. On average, the time to compute a Phase 1 bottommost tree is $0.9084s$ and that to compute a Phase 2 optimized tree is $1.7345s$. Out of the remaining 495 cases where the software fails to generate a tree, the average time to report failure is $0.8717s$.

5.3.2 Robustness Tests

We design robustness testing experiments to prove that our robust tree generator indeed generates more robust trees. By robust, we mean that when tested against a set of randomly generated weather instances, a more robust tree should have a higher probability of clearance. In the experiments, we use the same quadrant as in Section. 5.3.1.

Test Setup

In order to test robustness, instead of generating purely random weather ensembles, we simulate weather in real world scenarios where for a specific time point t_f, there are a set of weather forecasts, W_1, ..., W_N, each of which has a probability, $p(W_i)$, that it happens. The $p(W_i)$'s satisfy $\sum_{i=1}^{N} p(W_i) = 1$. When the time t_f approaches, the real weather condition is usually very similar to one of the forecasts W_i, but with slight difference, such as offsets in cell positions or changes of cell severities.

To better simulate real world weather scenarios, we first generate 3 sets of random *seed* weather cell positions, which correspond to 3 weather forecasts. The seed positions are a group of 3D vertices with cardinality $numCells$. We call them S_{w1}, S_{w2} and S_{w3}. The seeds are used to generate more weather instances for testings, which simulate the real weather condition at time t_f. We use a parameter $rangeFromSeed$ to specify the allowed range of cell offsets relative to their corresponding positions in S_{wi}. The range we use is $(0, rangeFromSeed)$. It is easy to see that the larger $rangeFromSeed$ is, the more the possible offset from the seed positions to their newly generated positions can be.

In the experiment, we compute trees in the quadrant populated with weather ensembles consisting of instances generated from one or more of the 1st, 2nd or 3rd seeds, and then test the generated trees against more of such seed-based weather instances. We compare the probabilities of clearance, p_t's, among the trees and expect that the trees generated with instances from all 3 seeds to have higher p_t's, hence to be the more robust ones.

We are also interested in testing another way to enhance robustness. The idea is simple and straightforward: To generate a robust tree with RNP requirement (lane width) rnp_s, we compute a tree with a larger RNP requirement rnp_l ($rnp_l > rnp_s$). The effect of using rnp_l is similar to that of adjusting a weather ensemble's $rangeFromSeed$. The meanings are different though. One is to shrink the widths of trees so that they can avoid more weather cells, while

the other is to gain robustness by limiting the space occupied by cells. We use $newRNPRatio$ to represent rnp_s/rnp_l ($newRNPRatio \leqslant 1$).

To conclude the parameters used in our robustness tests, besides the ones used in Section. 5.3.1, including the number of instances in an ensemble $numInstances$ (each instance has the same $1/numInstances$ probability of happening), the number of cells in an ensemble $numCells$, the width range of cells $cellWidth$, the height range of cells $cellHeight$, the deviation probability of a cell dev_p, the tree thickness measurement rnp, and the threshold probability criterion $p_{nodeEdge}$, we also have the following new parameters involved and adjustable: $rangeFromSeed$ and $newRNPRatio$. (In the experiments, the parameter $cellHeight$ is defaulted to a constant because it is essentially the same parameter as $numCells$. Also dev_p is defaulted to be 1 as explained in Section. 5.3.1.)

Instead of thoroughly testing all the combinations of parameters, we first conduct a careful pre-selection process that helps select the combinations with higher qualities, which measures the crowdedness of weather ensembles provided trees' rnps. The definition of the quality of a combination $comb_i$ = ($numInstances$, $numCells$, $cellWidth$, $p_{nodeEdge}$, $rangeFromSeed$, $newRNPRatio$), Q_{comb_i}, is as follows. Based on $comb_i$, we generate 1000 random weather instances, and then compute trees out of these instances. We call the number of successful cases n_s and Q_{comb_i} is defined to be $Q_{comb_i} = n_s/1000$. The pre-selection process evaluates the quality of each combination and only the ones satisfying $Q_{comb_i} >= 0.3$ are used in the experiments. The pre-selection step is necessary in that it still allows us to test most of the meaningful combinations while helping us avoid spending excessive amount of time on testing cases with overly crowded weather ensembles.

Test Results Analysis

We tested and analyzed over 10 million [tree, weather instance] pairs on our randomly generated data sets. The test results are shown in tables 5.1 and 5.2. Elements in the first column of each table are in the form of (i_1, i_2, i_3), which refers to a weather ensemble that has in total $(i_1 + i_2 + i_3)$ instances. And i_1 of the instances are generated from seed S_{w1}, i_2 of them are from seed S_{w2} and i_3 are from S_{w3}. We call such an ensemble an (i_1, i_2, i_3)-ensemble.

The tables present the average robustness of the trees generated with a specific combination of parameter values. For example, the number 0.5682 ($numCells = 5.0$, $(i_1, i_2, i_3) = (1, 0, 0)$) in Table 5.1 represents the test result

that on average, trees computed against an ensemble containing 1 weather instance generated from seed S_{w1}, consisting of 5 cells (other parameters are random), have a probability of 56.82% to be clear of weather instances that are randomly generated from one of the three seeds with the same set of parameters. The quick fact is that the larger the number in a cell is, the more robust the corresponding tree is.

We see that throughout the tables, the numbers in the first 3 rows in the same column are monotonically increasing, so are the numbers in the last 3 rows, which meets our expectations and testifies that with the same set of parameters, trees from weather instances generated based on more seeds are more robust than that from less seeds. One more fact is that in the same column, the value in the ith $(1 \leqslant i \leqslant 3)$ row is always less than that in the $(i+3)$th row, where the number of instances in the ensemble is 3 times larger. The explanation of the fact is that the more the number of instances generated from a seed S_{wi} are, the farther away that the tree is from S_{wi}'s positions. Hence when a new instance is generated from S_{wi}, the tree is more likely to be clear of that instance.

Further, we examined the robustness of trees with different $newRNPRatio$ values as shown in the last 4 columns of Table 5.2. Again, the results proves our conjecture that the trees generated with a larger $newRNPRatio = rnp_l > rnp_s$ are less robust than the ones with a smaller $newRNPRatio$. The tables show these additional stylized facts as well: (1) The larger $numCells$ is, the less robust the generated tree is; (2) Similarly, robustness decreases as $cellWidth$ increases; (3) When $rangeFromSeed$ increases, the weather instances generated are more random. Hence the robustness decreases.

We analyze the parameter $p_{nodeEdge}$ separately. All the tables report the experiment results when $p_{nodeEdge}$ is within the range of $[0.7, 0.1]$. But when $p_{nodeEdge}$ is smaller, the test results are mixed, where a tree computed from an $(1,1,1)$-ensemble may perform even worse than that from an $(1,1,0)$-ensemble or an $(1,0,0)$-ensemble. We explain the reason with an example: When $p_{nodeEdge} = 0.6$, every node or edge in a tree generated from an $(1,1,1)$-ensemble needs to be free of weather cells from only 2 of the 3 instances, because $\frac{2}{3} \approx 0.67 > 0.6$. But different edges or nodes in the same tree may be clear of different sets of instances. For example, edge e_i is clear of instances 1 and 2, while edge e_j is clear of instances 1 and 3. Consequently, when considered as a whole, the tree is only clear of part of each instance in the ensemble. Therefore, when tested against a new weather instance, the tree has a high probability to fail. Although all the nodes and edges in the tree are robust, the tree itself is

Parameters	$numCells$			$cellWidth$		
(i_1, i_2, i_3)-ensemble	5.0	8.0	11.0	3.0	5.5	8.0
(1, 0, 0)	0.5682	0.4417	0.3400	0.5314	0.4421	0.3765
(1, 1, 0)	0.6733	0.5624	0.4873	0.6272	0.5716	0.5242
(1, 1, 1)	0.7885	0.7019	0.6352	0.7478	0.7037	0.6742
(3, 0, 0)	0.6083	0.4942	0.4192	0.5679	0.5007	0.4531
(3, 3, 0)	0.6879	0.5778	0.4983	0.6357	0.5851	0.5434
(3, 3, 3)	0.8015	0.7245	0.6591	0.7546	0.7307	0.6998

Table 5.1: Robust tree test results(1). The number in each cell represents the robustness of trees generated from its corresponding values of a set of parameters. The larger the number is, the more robust the trees are. This table presents the results of testing parameters $numCells$ and $cellWidth$.

Parameters	$rangeFromSeed$			$newRNPRatio$			
(i_1, i_2, i_3)-ensemble	1.0	3.0	5.0	0.4	0.6	0.8	1.0
(1, 0, 0)	0.4894	0.4466	0.4139	0.4960	0.4631	0.4351	0.4037
(1, 1, 0)	0.6832	0.5575	0.4822	0.6435	0.6053	0.5574	0.4912
(1, 1, 1)	0.8425	0.6976	0.5855	0.7856	0.7396	0.6871	0.6246
(3, 0, 0)	0.5256	0.5101	0.4860	0.5606	0.5214	0.4922	0.4547
(3, 3, 0)	0.6822	0.5694	0.5124	0.6565	0.6121	0.5672	0.5164
(3, 3, 3)	0.8595	0.7180	0.6075	0.7996	0.7609	0.7137	0.6392

Table 5.2: Robust tree test results(2). This table presents the results of testing parameters $rangeFromSeed$ and $newRNPRatio$.

not a robust tree.

In order to avoid the problem, $p_{nodeEdge}$ must be large enough so that an edge or node needs to be clear of at least one weather instance generated from each seed. Specifically, suppose there are n seeds, S_{w1}, ..., S_{wn} and an $(i_1, ..., i_n)$-ensemble generated from the seeds, the instances of whom have the same probability of happening $1/\sum_{i=1}^{n}(i_i)$. Then based on the pigeonhole principle, $p_{nodeEdge}$ must satisfy $p_{nodeEdge} \geqslant 1 - (min(i_1, ..., i_n) - 1)/\sum_{i=1}^{n}(i_i)$.

To solve the problem, a straightforward method is that each edge or node stores the subset of weather instances that it is free of and the tree has all its edges and nodes clear of the instances generated from the same subset of seeds. But this is essentially the same as computing a tree directly from an ensemble of instances generated from less seeds. A better way to solve the problem and a possibly better definition of robust trees will be investigated in future research.

Chapter 6

Open Problems and Future Research

1. In Chapter 2, we proved that routing r red and b blue thick paths in a polygonal domain in the plane is NP-Hard. Further, we designed and implemented a heuristic using a bottommost lane packing technique. The hardness proof applies directly to the three-dimensional version of the problem. How can we extend the heuristic to three dimensions, where there is not a clear notion of "bottommost"? What other heuristics can be devised for this case?

2. Instead of routing red and blue thick paths as described in Chapter 2, we can consider the problem of routing the maximum number of paths, for a single class of aircraft, but with the constraint that the thick paths are disjoint and homotopically distinct (i.e., "threaded differently", in a topological sense).

3. In Chapter 4, we proved that it is NP-Hard to find a convex transversal for a set of (intersecting) 2D segments. What is the complexity of the problem for a set of *unit* length segments, a set of disks, or a set of pseudo-disks? Our conjecture is that it remains NP-Hard for these cases. It may be possible to extend our proof technique to these cases.

4. Also in Chapter 4, we proved that in 3D, it is NP-Hard to find a convex transversal for a set of balls. Can the proof be adapted to apply to *unit* balls? Our conjecture is that the problem remains NP-Hard for unit balls in 3D.

5. Still in Chapter 4, we gave a dynamic program to solve the case for disjoint segments, if the vertices of the transversal are from a given set of candidate positions. We do not know if the problem is solvable if the candidate set is not provided. This seems to be a challenging open problem.

6. In Chapter 5, we showed that to find a highly probable path for a set of (unit) segments is NP-Hard. The complexity is open, as far as we know, for the case of disks or unit disks in the plane. We are also interested in finding polynomial-time algorithms to solve some special cases of the problem, with different types of restrictions, e.g., when we require the path to be monotone,or we restrict the size of the domain so that the total number of substantially different disks is limited. Further, we are interested in realistic input models for sets of constraints that arise in our motivating application – hazardous weather avoidance. Sets of constraints that arise from an ensemble of weather forecasts are likely to show special structure, which may be captured in an appropriate model, allowing us to obtain efficient (provable) algorithms for these cases.

Bibliography

[1] *Report of the RTCA Board of Directors' Select Committee on Free Flight.* RTCA, Inc., Washington, DC, Jan. 1995.

[2] 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.

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

[4] 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.

[5] N. Aggarwal and K. Fujimura. Motion planning amidst planar moving obstacles. In *Proceedings of ICRA*, pages 2153–2158, 1994.

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

[7] D. Alberts and M.R. Henzinger. Average case analysis of dynamic graph algorithms. In *SODA '95*, pages 312–321, 1995.

[8] D. Alevras, M. Grotschel, and R. Wessali. Capacity and survivability models for telecommunication networks. Konrad-Zuse-Zentrum fur Informationstechnik, Berlin, Germany, 1997.

[9] H. Alt, S. Cabello, P. Giannopoulos, and C. Knauer. On some connection problems in straight-line segment arrangements. In *Abstracts of the 27th European Workshop on Computational Geometry (EuroCG 11)*, pages 27–30, 2011.

[10] N.M. Amato. An optimal algorithm for finding the separation of simple polygons. In *WADS '93: Proceedings of the Third Workshop on Algorithms and Data Structures*, pages 48–59, London, UK, 1993. Springer-Verlag.

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

[12] E. M. Arkin, R. Connelly, and J. S. B. Mitchell. On monotone paths among obstacles, with applications to planning assemblies. In *Proc. 5th Annual ACM Symposium on Computational Geometry*, pages 334–343, 1989.

[13] E. M. Arkin, C. Dieckmann, C. Knauer, J. S. B. Mitchell, V. Polishchuk, L. Schlipf, and S. Yang. Convex transversals. In Jörg Rüdiger Sack Frank Dehne, John Iacono, editor, *Proc. 12th Workshop Algorithms Data Struct.*, volume 6844 of *Lecture Notes Comput. Sci.*, pages 49–60. Springer-Verlag, 2011.

[14] E. M. Arkin, J. S. B. Mitchell, and V. Polishchuk. Maximum thick paths in static and dynamic environments. *Computational Geometry: Theory and Applications*, 43(3):279–294, 2010.

[15] E. M. Arkin, J. S. B. Mitchell, V. Polishchuk, and S. Yang. Convex transversals. In *Fall Workshop on Computational Geometry*, 2010.

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

[17] S. Bae, J. Kim, and K. Chwa. Optimal construction of the city voronoi diagram. In *ISAAC*, pages 183–192, 2006.

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

[19] S. Basu, R. Pollack, and M. Roy. On computing a set of points meeting every cell defined by a family of polynomials on a variety. *J. Complex.*, 13(1):28–37, 1997.

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

[21] S. Bereg and D.G. Kirkpatrick. Approximating barrier resilience in wireless sensor networks. In *5th ALGOSENSORS*, volume 5804 of LNCS, pages 29–40. Springer, 2009.

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

[23] 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.

[24] A. Bley. On the complexity of vertex-disjoint length-restricted path problems. *Computational Complexity*, 12(3-4):131–149, 2003.

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

[26] J. 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.

[27] M. Brennan. Airspace flow programs - a fast path to deployment. *Journal of Air Traffic Control*, 49(1), 2007.

[28] H. Brönnimann and M.T. Goodrich. Almost optimal set covers in finite vc-dimension. *Discrete & Computational Geometry*, 14(4):463–479, 1995.

[29] R. L. Brooks. On colouring the nodes of a network. *Proc. Cambridge Phil. Soc*, 37:194–197, 1941.

[30] 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.

[31] J. Canny, A. Rege, and J. Reif. An exact algorithm for kinodynamic planning in the plane. *Discrete Comput. Geom.*, 6:461–484, 1991.

[32] J. Canny and J. H. Reif. New lower bound techniques for robot motion planning problems. In *Proc. 28th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 49–60, 1987.

[33] R. D. Carr, S. Doddi, G. Konjevod, and M. Marathe. On the red-blue set cover problem. In *11th Annual ACM-SIAM Symposium on Discrete algorithms*, pages 345–353, 2000.

[34] W. Chan, M. Refai, and R. DeLaura. Validation of a model to predict pilot penetrations of convective weather. In *AIAA Aviation, Technology, Integration and Operations Conf.*, Belfast, Northern Ireland, Sept. 2007.

[35] C. Chekuri. Multicommodity flow, well-linked terminals and routing problems. In Ming-Yang Kao, editor, *Encyclopedia of Algorithms*. Springer, 2008.

[36] C. Chekuri, S. Khanna, and F. B. Shepherd. Edge-disjoint paths in planar graphs. In *FOCS '04: Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS'04)*, pages 71–80, Washington, DC, USA, 2004. IEEE Computer Society.

[37] A. Chen, S. Kumar, and T. H. Lai. Designing localized algorithms for barrier coverage. In *13th Annual ACM International Conference on Mobile Computing and Networking (MobiCom'07)*, pages 63–74, 2007.

[38] A. Chen, T. Lai, and D. Xuan. Measuring and guaranteeing quality of barrier-coverage in wireless sensor networks. In *9th ACM international symposium on Mobile ad hoc networking and computing*, pages 421–430, New York, NY, USA, 2008.

[39] 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.

[40] Y.-J. Chiang, J.T. Klosowski, C. Lee, and J.S.B. Mitchell. Geometric algorithms for conflict detection/resolution in atm. In *IEEE Conf. on Decision and Control*, San Diego, CA, 1997.

[41] 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.

[42] 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.

[43] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 2009.

[44] P. Crescenzi and V. Kann. A compendium of NP optimization problems. In *Complexity and Approximation. Combinatorial Optimization Problems and Their Approximability Properties*. Springer, Berlin, 1999.

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

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

[47] R. DeLaura and J. Evans. An exploratory study of modeling en route pilot convective storm flight deviation behavior. In *12th American Meteorological Society Conf. on Aviation, Range, and Aerospace Meteorology*, Atlanta, GA, Jan./Feb. 2006.

[48] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

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

[50] D. P. Dobkin, D. L. Souvaine, and C. J. Van Wyk. Decomposition and intersection of simple splinegons. *Algorithmica*, 3:473–486, 1988.

[51] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 1999.

[52] 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.

[53] A. Efrat, A. Itai, and M.J. Katz. Geometry helps in bottleneck matching and related problems. *Algorithmica*, 31(1):1–28, 2001.

[54] 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.

[55] 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.

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

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

[58] A. Frank. Packing paths, cuts, and circuits — a survey. In B. Korte, L. Lovasz, H.J. Promel, and A. Schrijver, editors, *Paths, Flows, and VLSI Layout*, pages 49–100. Berlin: Springer-Verlag, 1990.

[59] K. Fujimura. Motion planning amid transient obstacles. *The International Journal of Robotics Research*, 13(5):395–407, 1994.

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

[61] K. Fujimura and H. Samet. Planning a time-minimal motion among moving obstacles. *Algorithmica*, 10(1):41–63, 1993.

[62] H.N. Gabow, S.N. Maheswari, and L.J. Osterweil. On two problems in the generation of program test paths. *IEEE Trans. Software Eng.*, 2(3):227–231, 1976.

[63] 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.

[64] 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.

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

[66] M. Gibson, G. Kanade, and K. Varadarajan. On isolating points using disks.

[67] J. Glimm and D.H. Sharp. Complex fluid mixing flows: Simulation vs. theory vs. experiment. *SIAM News*, 39(5), June 2006.

[68] M.T. Goodrich and J. Snoeyink. Stabbing parallel segments with a convex polygon. *Comput. Vision Graph. Image Process.*, 49(2):152–170, 1990.

[69] R.L. Graham, B.D. Lubachevsky, K.J. Nurmela, and P.R. J. Östergård. Dense packings of congruent circles in a circle. *Discrete Mathematics*, 181(1-3):139–154, 1998.

[70] 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.

[71] V. Guruswami, S. Khanna, R. Rajaraman, B. Shepherd, and M. Yannakakis. Near-optimal hardness results and approximation algorithms for edge-disjoint paths and related problems. pages 19–28, 1999.

[72] M. Harris, A. Raza, Z. Rojas, E. Price, and S. Rajlawat. Preliminary design analysis of dynamic airspace super sectors DASS, 2006.

[73] H. Hering. Air traffic freeway system for Europe. Report EEC Note No. 20/05, EUROCONTROL Experimental Centre, 2005. www.eurocontrol.int.

[74] J. Hershberger and J. Snoeyink. Computing minimum length paths of a given homotopy class. *Comput. Geom. Theory Appl.*, 4:63–98, 1994.

[75] J. Hershberger and S. Suri. An optimal algorithm for Euclidean shortest paths in the plane. *SIAM J. Comp.*, 28:2215–2256, 1999.

[76] 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.

[77] H. van der Holst and J. C. de Pina. Length-bounded disjoint paths in planar graphs. *Discr. Appl. Math.*, 120(1-3):251–261, 2002.

[78] J. Hopcroft and R. M. Karp. An $n^{(5/2)}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2(4):225–231, 1973.

[79] C. Hsu. General river routing algorithm. In *Proceedings of the twentieth design automation conference on Design automation*, pages 578–583, 1983.

[80] T. C. Hu. *Integer programming and network flows*. Addison-Wesley, Reading, MA, 1969.

[81] T.C. Hu, A.B. Kahng, and G. Robins. Solution of the discrete plateau problem. *Proc. Natl. Acad. Sci.*, pages 9235–9236, October 1992.

[82] T.C. Hu, A.B. Kahng, and G. Robins. Optimal robust path planning in general environments. *IEEE Transactions on Robotics and Automation*, 9:775–784, 1993.

[83] M. Iri. *Survey of mathematical programming*. (A. Prékopa, Ed.) North-Holland, Amsterdam, Netherlands, 1979.

[84] A. Itai and Y. Shiloach. Maximum flow in planar networks. *SIAM J. Comput.*, 8(2):135–150, May 1979.

[85] S. Iwanowski. Testing approximate symmetry in the plane is NP-hard. *Theor. Comput. Sci.*, 80(2):227–262, 1991.

[86] H. Kaplan, N. Rubin, and M. Sharir. Line transversals of convex polyhedra in $\mathbb{R}^3$. In *SODA '09*, pages 170–179, 2009.

[87] R. M. Karp. On the computational complexity of combinatorial problems. 5:45–68, 1975.

[88] L. E. Kavraki, P. Švestka, J.-C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high dimensional configuration spaces. *IEEE Trans. Robot. Autom.*, 12:566–580, 1996.

[89] 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.

[90] J. Kim, J. S. B. Mitchell, V. Polishchuk, S. Yang, and J. Zou. Routing multi-class traffic flows in the plane. *Computational Geometry: Theory and Applications*, 45:99 – 114, Apr. 2012.

[91] A. Klein, L. Cook, B. Wood, and D. Simenauer. Airspace capacity estimation using flows and weather-impacted traffic index. In *Integrated Communications, Navigation and Surveillance (ICNS'08)*, pages 1–12, 2008.

[92] S. Kloder and S. Hutchinson. Barrier coverage for variable bounded-range line-of-sight guards. In *ICRA*, pages 391–396. IEEE, 2007.

[93] 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).

[94] 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.

[95] S.G. Kolliopoulos and C. Stein. Approximating disjoint-path problems using greedy algorithms and packing integer programs. *Lecture Notes in Computer Science*, 1412:153–168, 1998.

[96] M. R. Kramer and J. van Leeuwen. Wire-routing is NP-complete. Technical report RUU-CS-82-4, Department of Computer Science, University of Utrecht, 1982.

[97] S. Krishna and J. Krozel. Impact analysis for in-flight icing hazards. In *AIAA Guidance, Navigation, and Control Conf.*, Chicago, IL, Aug. 2009.

[98] J. Krozel, R. Jakobovits, and S. Penny. An algorithmic approach for airspace flow programs. *Air Traffic Control Quarterly*, 14(3), 2006.

[99] J. Krozel, W. McNichols, J. Prete, and T. Lindholm. Causality analysis for aviation weather hazards. In *AIAA Aviation Technology, Integration, and Operations Conf.*, Anchorage, AK, Sept. 2008.

[100] J. Krozel, W. McNichols, J. Prete, and T. Lindholm. Causality analysis for aviation weather hazards. In *AIAA Aviation Technology, Integration, and Operations (ATIO) Conference*, Anchorage, AK, Sept. 2008.

[101] J. Krozel, J. S. B. Mitchell, V. Polishchuk, and J. Prete. Airspace capacity estimation with convective weather constraints. In *AIAA Guidance, Navigation, and Control Conference*, Aug 2007.

[102] J. Krozel, J.S.B. Mitchell, V. Klimenko, T. Lindholm, J. Prete, N. Downs, and S. Krishna. Translation of weather information to traffic flow management (TFM) impact: Weather translation model for TFM decisions including FACET demonstrations. Technical Report 34N0707-002-R0, Metron Aviation, October 2008.

[103] J. Krozel, J.S.B. Mitchell, V. Polishchuk, and J. Prete. Maximum flow rates for capacity estimation in level flight with convective weather constraints. *Air Traffic Control Quarterly*, 15(3), 2007.

[104] J. Krozel, J.S.B. Mitchell, V. Polishchuk, and J. Prete. Maximum flow rates for capacity estimation in level flight with convective weather constraints. *Air Traffic Control Quarterly*, 15(3):209–238, 2007.

[105] J. Krozel and Jr. Murphy, J.T. Weather hazard requirements for ngats aircraft. In *Integrated Communications, Navigation, and Surveillance Conf.*, Herndon, VA, May 2007.

[106] J. Krozel, S. Penny, J. Prete, and J.S.B. Mitchell. Automated route generation for avoiding deterministic weather in transition airspace. *Journal of Guidance, Control, and Dynamics*, 30(1):144–153.

[107] 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.*, August 2004.

[108] J. Krozel, J. Prete, J. S. B. Mitchell, Joondong Kim, and Jason Zou. Capacity estimation for super-dense operations. In *AIAA Guidance, Navigation, and Control Conf.*, Aug 2008.

[109] J. Krozel, J. Prete, J.S.B. Mitchell, J. Kim, and J. Zou. Capacity estimation for super-dense operations. In *AIAA Guidance, Navigation, and Control Conf.*, Honolulu, HI, Aug. 2008.

[110] J. Krozel, J. Prete, J.S.B. Mitchell, J. Kim, and J. Zou. Capacity estimation for super-dense operations. In *AIAA Guidance, Navigation, and Control Conference*, 2008.

[111] J. Krozel, J. Prete, J.S.B. Mitchell, P. Smith, and A.D. Andre. Designing on-demand coded departure routes. In *AIAA Guidance, Navigation, and Control Conference*, 2006.

[112] K. Kuhn. Analysis of thunderstorm effects on aggregated aircraft trajectories. *Journal of Aerospace Computing, Information and Communication*, 5, April 2008.

[113] S. Kumar, T.-H. Lai, and A. Arora. Barrier coverage with wireless sensors. *Wireless Networks*, 13(6):817–834, 2007.

[114] Y. Kusakari, H. Suzuki, and T. Nishizeki. Finding a shortest pair of paths on the plane with obstacles and crossing areas. In J. Staples et al., editor, *Algorithms and Computation*, pages 42–51. Springer, Berlin,, 1995.

[115] J.-C. Latombe. *Robot Motion Planning*. Kluwer, Boston, 1991.

[116] S.M. Lavalle. *Planning Algorithms*. Cambridge University Press, New York, NY, 2006.

[117] D. T. Lee, C. D. Yang, and C. K. Wong. Rectilinear paths among rectilinear obstacles. *Discr. Appl. Math.*, 70:185–215, 1996.

[118] 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.

[119] Z. Li and J. F. Canny, editors. *Nonholonomic Motion Planning*. Kluwer, Norwell, MA, 1992.

[120] T. Lindholm, J. Krozel, and J. S. B. Mitchell. Concept of operations for addressing multiple types of en route hazardous weather constraints in nextgen. In *AIAA Aviation Technology, Integration, and Operations Conf.*, Hilton Head, SC, Sept. 2009.

[121] 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, October 1995.

[122] M. Löffler and M.J. van Kreveld. Largest and smallest convex hulls for imprecise points. *Algorithmica*, 56(2):235–269, 2010.

[123] J. F. Lynch. The equivalence of theorem proving and the interconnection problem. *ACM SIGDA Newsletter*, 5:31–65, 1975.

[124] F. M. Maley. *Single-Layer Wire Routing and Compaction*. MIT Press, Cambridge, MA, 1990.

[125] 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.

[126] D. Michalek and H. Balakrishnan. Identification of robust routes using convective weather. In *Eighth USA/Europe Air Traffic Management Research and Development Seminar (ATM)*, 2009.

[127] J.S.B. Mitchell. On maximum flows in polyhedral domains. *Journal of Computer and System Sciences*, 40:88–123, 1990.

[128] J.S.B. Mitchell. A new algorithm for shortest paths among obstacles in the plane. *Ann. Math. Artif. Intell.*, 3:83–106, 1991.

[129] J.S.B. Mitchell. L_1 shortest paths among polygonal obstacles in the plane. *Algorithmica*, 8:55–88, 1992.

[130] J.S.B. Mitchell. Shortest paths and networks. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 24, pages 445–466. CRC Press LLC, 1997.

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

[132] 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.

[133] J.S.B. Mitchell and V. Polishchuk. Thick non-crossing paths and minimum-cost flows in polygonal domains. In 2*3rd ACM Symposium on Computational Geometry*, pages 56–65, 2007.

[134] J.S.B. Mitchell, V. Polishchuk, and J. Krozel. Airspace throughput analysis considering stochastic weather. In *AIAA Guidance, Navigation, and Control Conference*, Aug 2006.

[135] J.S.B. Mitchell and M. Sharir. New results on shortest paths in three dimensions. In *SCG '04: Proceedings of the twentieth annual symposium on Computational geometry*, pages 124–133, New York, NY, USA, 2004. ACM Press.

[136] N.H. Mustafa and S. Ray. Ptas for geometric hitting set problems via local search. In *In Symposium on Computational Geometry(SoCG)*, pages 17–22, 2009.

[137] D. Nieuwenhuisen, J.P. van den Berg, and M.H. Overmars. Efficient path planning in changing environments. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems - IROS'07*, 2007.

[138] A. Orda and R. Rom. On continuous network flows. *Operations Research Letters*, 17, February 1995.

[139] J.B. 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.

[140] J.B. Orlin, 2007. Personal communication.

[141] E. Papadopoulou. Personal communication.

[142] E. Papadopoulou. k-pairs non-crossing shortest paths in a simple polygon. *Int. J. Comp. Geom. Appl.*, 9(6):533–552, 1999.

[143] V. Polishchuk. *Thick Non-Crossing Paths and Minimum-Cost Continuous Flows in Geometric Domains*. PhD thesis, Stony Brook University, Aug 2007. Available at `http://cs.helsinki.fi/~polishch/pages/thesis.pdf`.

[144] J. Prete. *Aircraft Routing in the Presence of Hazardous Weather*. PhD thesis, Computer Science, Stony Brook University, Stony Brook.NY., 2007.

[145] J. Prete, J. Krozel, J. S. B. Mitchell, Joondong Kim, and Jason Zou. Flexible, performance-based route planning for super-dense operations. In *AIAA Guidance, Navigation, and Control Conf.*, Aug 2008.

[146] 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.*, August 2004.

[147] J. Reif and M. Sharir. Motion planning in the presence of moving obstacles. *J. ACM*, 41(4):764–790, July 1994.

[148] 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.

[149] D. Richards. Complexity of single-layer routing. *IEEE Trans. Computers*, 33(3):2860–288, 1984.

[150] N. Robertson and P. D. Seymour. Graph minors. xiii: the disjoint paths problem. *J. Comb. Theory Ser. B*, 63(1):65–110, 1995.

[151] 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.

[152] D. K. Schmidt. On modeling atc work load and sector capacity. *Journal of Aircraft*, 13(7), 1975.

[153] Y. Schreiber. Shortest paths on realistic polyhedra. In *SCG '07: Proceedings of the twenty-third annual symposium on Computational geometry*, pages 74–83, New York, NY, USA, 2007. ACM Press.

[154] Y. Schreiber and M. Sharir. An optimal-time algorithm for shortest paths on a convex polytope in three dimensions. In *SCG '06: Proceedings of the twenty-second annual symposium on Computational geometry*, pages 30–39, New York, NY, USA, 2006. ACM Press.

[155] R. Sharman, C. Tebaldi, G. Wiener, and J. Wolff. An integrated approach to mid- and upper-level turbulence forecasting. *Weather and Forecasting*, 21:268–287, 2006.

[156] Y. Shiloach. A polynomial solution to the undirected two paths problem. *J. ACM*, 27(3):445–456, 1980.

[157] L. Song, C. Wanke, and D. Greenbaum. Predicting sector capacity for tfm decision support. In *AIAA 6th Technology, Integration, and Operations Conf.*, Wichita, KS, Sept. 2006.

[158] L. Song, C. Wanke, and D. Greenbaum. Predicting sector capacity under severe weather impact for traffic flow management sector capacity under severe weather impact for traffic flow management. In *AIAA Aviation Technology, Integration, and Operations Conf.*, Belfast, Northern Ireland, Sept. 2007.

[159] L. Song., C. Wanke, D. Greenbaum, S. Zobell, and C. Jackson. Methodologies for estimating the impact of severe weather on airspace capacity. In *26th Intern. Congress of the Aeronautical Sciences*, Anchorage, AK, Sept. 2008.

[160] 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.

[161] G. Strang. Maximal flow through a domain. *Math. Program.*, 26:123–143, 1983.

[162] H. Swenson, R. Barhydt, and M. Landis. Next generation air transportation system (NGATS) air traffic management (ATM)-airspace project. Tech. Report, Version 6.0, NASA, 2006.

[163] J. Takahashi, H. Suzuki, and T. Nishizeki. Finding shortest non-crossing rectilinear paths in plane regions. In *ISAAC*, pages 98–107, 1993.

[164] A. Tamir. Problem 4-2 (New York University, Dept. of Statistics and Operations Research), Problems Presented at the Fourth NYU Computational Geometry Day (3/13/87).

[165] E. Tardos. A strongly polynomial minimum cost circulation algorithm. *Combinatorica*, 5(3):247–255, 1985.

[166] K.-C. R. Tseng. Resilience of wireless sensor networks. Master's thesis, The University Of British Columbia (Vancouver), April 2011.

[167] J.P. van den Berg. *Path Planning in Dynamic Environments*. PhD thesis, Utrecht University, 2007.

[168] J.P. van den Berg, D. Ferguson, and J. Kuffner. Anytime path planning and replanning in dynamic environments. In *ICRA*, pages 2366–2371, 2006.

[169] J.P. van den Berg, D. Nieuwenhuisen, L. Jaillet, and M.H. Overmars. Creating robust roadmaps for motion planning in changing environments. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems - IROS'05*, pages 2415–2421, 2005.

[170] J.P. van den Berg and M.H. Overmars. Prioritized motion planning for multiple robots. In I*EEE/RSJ Int. Conf. on Intelligent Robots and Systems - IROS'05*, pages 2217–2222, 2005.

[171] J.P. van den Berg and M.H. Overmars. Roadmap-based motion planning in dynamic environments. *IEEE Transactions on Robotics*, 21(5):885–897, 2005.

[172] J.P. van den Berg and M.H. Overmars. Planning the shortest safe path amidst unpredictably moving obstacles. In *Proc. Workshop on Algorithmic Foundations of Robotics - WAFR'06*, 2006.

[173] J.P. van den Berg and M.H. Overmars. Kinodynamic motion planning on roadmaps in dynamic environments. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems - IROS'07*, 2007.

[174] J. Vygen. Disjoint paths. Research Institute for Discrete Mathematics, University of Bonn, Report No. 94816, 1998.

[175] 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.

[176] 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.

[177] C. D. Yang, D. T. Lee, and C. K. Wong. The smallest pair of noncrossing paths in a rectilinear polygon. *IEEE Trans. Comput.*, 46(8):930–941, 1997.

[178] S. Yang, J. Kim, J. Krozel, J. S. B. Mitchell, V. Polishchuk, and J. Zou. Flexible airlane generation to maximize flow under hard and soft constraints. *Air Traffic Control Quarterly*, 19(2):1–26, 2011.

[179] 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.