Geometric Algorithms for Capacity Estimation and Routing in Air Traffic Management

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

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

Stony Brook University
August 2010

Stony Brook University
The Graduate School

Jingyu Zou

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

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

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

Professor Xiangmin Jiao
Department of Applied Mathematics and Statistics

Professor Jie Gao
Department of Computer Science

This dissertation is accepted by the Graduate School

Lawrence Martin
Dean of the Graduate School

Abstract of the Dissertation

Geometric Algorithms for Capacity Estimation and Routing in Air Traffic Management

by

Jingyu Zou

Doctor of Philosophy

in

Applied Mathematics and Statistics

(Operations Research)

Stony Brook University

2010

We study various problems that arise from two inter-related fundamental computational problems in Air Traffic Management (ATM): that of estimating the capacity of an airspace that is cluttered with hazardous weather constraints, and that of routing aircraft on trajectories in space-time to avoid hazardous weather constraints and to achieve maximum efficiency while maintaining safe separation distances between all pairs of aircraft at all times.

We model the problem of capacity estimation as a geometric problem of finding the maximum number of pairwise disjoint "thick" paths in a polygonal domain with holes. For the case where all thick paths are of the same width, we give a 1/2-approximation algorithm using geometric spanners. We also show experimentally that using a spanner (e.g., Delaunay graph) yields approximation ratios very close to one. For the case where thick paths are of two or more different widths, we show that the problem becomes strongly NP-hard; we also give polynomial-time algorithm for the special case where the order of the widths of the paths as they appear on the polygonal boundary is given.

We also study the capacity estimation problem for any airspace under different dominant demand (flow) directions and see how the directional capacity changes as a function of the demand direction. We lay out grids on the whole National Airspace System (NAS), put square or circular kernels of appropriate sizes on each grid point, and apply the directional capacity analysis for each kernel; this way we get an idea on the directional capacity for the whole NAS. Further, we describe how the grid-based analysis can be used in ATM applications.

We investigate methods of computing flexible trees of arrival and departure routes for the transition airspace. We model the problem of routing flows of arrival aircraft on a merge tree with constraints imposed by hazardous weather, special use airspace, and operational constraints on merge points. We have proved that this problem is in general NP-hard; an polynomial-time algorithm is presented to compute an optimal merge tree if the number of "layers" in the search graph is constant. The algorithm applies both to the case of fixed (constant) Required Navigation Performance (RNP) and the case of a mixture of different RNP values. We show experimental results demonstrating the application of the algorithm to problem instances that are based on actual weather data.

Contents

List of Figures

List of Tables

Acknowledgements

This dissertation could not have been completed without the guidance and support from my adviser, Professor Joseph S. B. Mitchell. I thank him for introducing me to the field of computational geometry, which I knew very little about before coming to Stony Brook but found fascinating about later on because of him. I thank him for welcoming me to his research group and for providing me with the unique and exceptional research atmosphere — one that is the most friendly, collaborative, supportive, creative and inspiring that I have ever imagined; it has always been fun to study and work under his guidance and in his group. To me, Joe is both a mentor and a friend, who has been caring and understanding in both academic matters and beyond. I am certain that the experience with him will be invaluable for my life.

I thank the faculty members who has served as referees in various stages of my PhD study (Preliminary Examination, Dissertation Defense, etc.): Professor Esther M. Arkin, Professor Jie Gao, Professor Xiangmin Jiao, Professor Joseph S. B. Mitchell and Professor Alan Tucker.

Further, I thank Professor Xiaolin Li for accepting and welcoming me into the department of applied mathematics and statistics at Stony Brook, for helping me adapt to the academic and social life of America, and for his help on lots of other personal things. I thank Professor Michael A. Bender for his suggestions and guidance in helping me make the transition from learning into the research phase early on during my PhD study. I thank all professors who taught me at Stony Brook (especially Joe, Estie, Jie, Michael, Professor Xianfeng Gu, Professor Yair Tauman, Professor John Pinezich) and the courses they taught were equally instrumental in broadening my view and inspiring my interest in research.

I am thankful to Dr. Rafal Kicinger, Dr. Jimmy Krozel and Dr. Joseph Prete of Metron Aviation, Inc. for insightful discussions that has greatly benefited my work. The weekly teleconferences with them have motivated and helped the investigations of many of the problems studied in this work.

I would like to thank my colleagues whom I have worked with at Stony Brook, in particular Joondong Kim, Irina Kostitsyna, Yinghua Li, Eli Packer, Valentin Polishchuk, Girishkumar R. Sabhnani, Rik Sarkar, and Shang Yang. The blackboard discussions and collaborations with them have always been fun and productive and haven been crucial to my PhD research.

I am indebted to my wife and my parents for constant encouragement and support, for their love and for everything. This work is dedicated to them.

1 Introduction

This thesis addresses two fundamental computational problems in Air Traffic Management (ATM): that of estimating the capacity of an airspace that is cluttered with hazardous weather constraints, and that of routing aircraft on trajectories in space-time to avoid hazardous weather constraints and to achieve maximum efficiency while maintaining safe separation distances between all pairs of aircraft at all times. These two problems are inter-related: we need to estimate the capacity of an airspace ahead of time in order to plan for future traffic to be routed up to but not exceeding the estimated capacity, while throughput analysis from an optimal routing of aircraft along trajectories can result in a better understanding of the actual airspace capacity.

Capacity Estimation

The “capacity” of an airspace is informally defined as the maximum number of aircraft per unit time that can utilize the airspace, given the various airspace *constraints*, which include restricted “no-fly” airspaces, such as Special Use Airspace (SUA), and regions of hazardous convective weather, turbulence, icing potential, etc.

Our investigations are done under operational assumptions considered to be in line with the Next Generation Air Transportation System (NextGen) [21], with a time frame of applicability of roughly 2020 to 2025, depending on the rapidness of implementation of certain technological advances in navigation and communication. We consider the estimation of capacity in a future ATM system where aircraft are not required to follow jet routes, as they generally are today (with the exception of a limited number of “free flight” aircraft at restricted altitudes). Rather, we consider operational rules for “en route” (between major airports) and “transition airspace” (in the local vicinity of major airports) where traffic finds routes through the gaps in airspace constraints. We generally assume that all aircraft have the same speed and follow the same Miles-In-Trail (MIT) standard, i.e., that aircraft abide by a common required separation along a “flow” of aircraft traveling single file on a common route. Under these assumptions, the maximum number of aircraft per unit time that enter an airspace, on their way across it, is proportional to the maximum number of non-interfering air lanes that can be packed in the airspace, while avoiding constraints and satisfying lateral separation standards between aircraft on different lanes. This is, generally, the definition of airspace capacity used throughout this thesis.

Capacity estimation plays an important role in aircraft route planning and scheduling. If the demand for an airspace, which is typically a *sector* in en route airspace or a transition airspace centered on a major airport, exceeds its estimated capacity, a Traffic Flow Management (TFM) strategy must be implemented. A TFM strategy employs a Ground Delay Program (GDP) to delay aircraft that are still on the ground, uses speed control and holding patterns to delay aircraft that are already airborne, and reroutes aircraft to a different, less obstructed, route, as feasible, given the constraints. Much research has been done with the purpose of minimizing the delay and rerouting costs of a TFM strategy, often using mathematical programming techniques; see, e.g., [4, 5].

Related Work Early work by Schmidt [43] investigated the traffic variables, routes, sector geometry, and control procedures that contribute to a control "difficulty index" that empirically quantifies the workload required on the part of the air traffic control team to manage a sector; the sector capacity can then be limited in such a way that the total workload, as measured by the control difficulty index, stays below a specified threshold. Wanke et al. [53] and Song et al. [45] studied the problem of predicting sector capacity by identifying a capacity value for each primary traffic flow pattern and predicting capacity under future flow patterns by matching the pattern with the most similar primary flow pattern; they defined "capacity" empirically as the number of aircraft simultaneously in the sector such that a sector performance index transitions from route structure to congestion at that point.

One problem central to our model of airspace capacity estimation is maximum flows in polygonal domains. This problem is a continuous and geometric analog of the celebrated max-flow/min-cut theorem from network analysis [1]. Strang [49] first studied this problem and proved the max-flow/min-cut theorem in the geometric setting. Mitchell [34] studied this problem from an algorithmic point of view and gave efficient algorithms to compute maximum flows. Krozel et al. [28, 29] were the first to apply the continous max-flow/min-cut theory to the ATM domain; they defined the theoretical upper bound on airspace capacity as the maximum number of air lanes that can be packed amidst obstacles and uses the aforementioned theory to compute capacity estimations under different flow organizations. The justification for this metric of capacity is that, assuming constant and common speeds and fixed separation (Miles in Trail) along a flow, the maximum number of aircraft per unit time that can travel through a sector, entering through the "source" portion of the boundary and exiting through the "sink" portion of the boundary is proportional to the number of air lanes that can be routed, as given by the max-flow/min-cut computation. Song et al. [46, 47] also used the max-flow/min-cut theory to compute an "available capacity ratio" for sectors.

Routing Aircraft Flows

In today's ATM system, aircraft usually follow fixed routes and do not have much flexibility in adjusting to changes in weather conditions and other constraints. The impact of hazardous weather on air traffic operations in transit and terminal airspace is significantly greater than it is on en route airspace and is a major cause of flight delays and cancellations in airspace around airports. Thus, effective route planning algorithms are needed in order to optimize the utilization and efficiency of airspace in the face of weather hazard constraints.

In the current air transportation system, the airspace around major airports is typically partitioned between arrival and departure traffic flows, organized in ways to allow for the safe and efficient flow of traffic to and from the terminal area. Today, traffic obeying Instrument Flight Rules (IFR) follow tree-like structures used for Standard Instrument Departures (SIDs) as well as Standard Terminal Arrival Routes (STARs). SIDs and STARs are based on fixed airway structures, defined by fixed navaids (fixed in location for all time) and fixed routes connecting navaids. Pilots, air traffic controllers, and their decision support systems use these well-defined SIDs and STARs as a basis for the safe and efficient routing of traffic on departures and on arrival routes. Arrival routes have a tree-like structure, typically with at most two routes coming together at a merge node, for ease of control. The use of SIDs and STARs adds to the predictability (and thus helps to lower controller workload), in that the SIDs and STARs are reused day after day and by most aircraft in a flow during the day.

One major drawback to using such a fixed routing structure in transition airspace ATM is that when disturbances due to hazardous weather appear, the routing structures are not easily adaptable to such constraints and, as a result, their benefits diminish. At times when hazardous weather crosses over the SIDs and STARs, pilots request deviations around the hazardous weather cells, passing to the left or right as the pilot prefers. The deviation from the structured routing disrupts the traffic flow organization, thereby deteriorating the predictability needed for safe oversight by air traffic controllers. For smaller weather hazards, this results in increased MIT between aircraft along a flow; such MIT restrictions are imposed by air traffic controllers in order to add to the "wiggle room" needed between aircraft in the flow and to increase flexibility in those cases in which hazardous weather constraints become large enough that portions of the structured routing become unusable, temporarily closing the air lanes. As a result, each of these circumstances reduces the traffic flow rate, often back-propagating into other ATM problems elsewhere in the National Airspace System (NAS).

In this thesis, we consider an alternative approach to transition airspace routing, based on a flexible routing structure suitable for NextGen. We retain the basic traffic flow structure organized in terms of an arrival (or departure) tree, however, we allow the fundamental navaids defining the tree to move around in space and time in order to adjust the tree structure so that it does not ever overlap hazardous weather constraints (if possible).

In the modeling of the problem, we consider a key parameter to be the Required Navigation Performance (RNP) for aircraft on the tree for a given period of time; the RNP is a distance that specifies how closely an aircraft is able to follow a specified lane centerline. (Aircraft with small RNP capability have better navigational and control equipment, allowing them to follow a prescribed route much more precisely than an aircraft with a large RNP performance parameter.) Another parameter in our model specifies the minimum allowed distance between two consecutive *merge points* along an arrival path in the merge tree. For control purposes, merge points where two flows come together must be separated by at least some specified distance; otherwise, multiple (three or more) flows effectively come together in the vicinity of a single point.

Advanced datalink communications and Area Navigation (RNAV) capabilities, such as low RNP parameters, are expected to be available in NextGen and will allow for flexible routing structures to be designed and coordinated "on the fly" while aircraft are en route. Network Centric Operations (NCO) will simultaneously communicate the tree structures to pilots, controllers, and dispatchers in the NAS. With a more flexible transition airspace routing solution, we expect that MIT restrictions will be reduced or eliminated, routes will maintain their tree-like structure even in the presence of weather constraints, and routes will remain open while weather constraints pass through the transition airspace. Thus, we expect that such flexible routing structures will be an enabling feature for uninterrupted Super Dense Operations in the terminal area, a goal for NextGen.

Related Work In general, the fundamental problem of weather avoidance routing for individual aircraft has been studied by many researchers. Optimal path planning algorithms are often based on grid search methods [11, 27, 31, 32, 33]. An alternative problem formulation is in terms of the *weighted region problem*, which has been studied in computational geometry and applied to air traffic routing [31, 35, 41]. A fundamental observation is that the local optimality criterion for an optimal route is that it obeys Snell's Law of Refraction as it passes between two neighboring regions of distinct weights (cost per unit distance). A further enhancement to the model for the ATM environment specifies a maximum number, k, of turns allowed for a polygonal path through a weather forecast map [27]; the premise of this model is that realistic routes should have a small number of turns (waypoints), in order to bound the workload of the pilot and controller required to track the solution. In general, these approaches mostly address the routing of a single aircraft rather than designing routes for multiple flows of aircraft, as addressed in this thesis.

Transition airspace routing essentially requires a systematic way to "disperse" departure traffic away from airport departure fixes or "funnel" arrival traffic into airport metering fixes. Historically, this has been achieved through SIDs for departures and STARs for arrivals (as discussed above). For the sake of discussion in this thesis, we concentrate on the arrival traffic funneling in to the airport, from an outer range ring into a "metering fix", which is a point located close to the airport, on the boundary of what is known as the *terminal airspace*. (The case of departure trees can be handled analogously, using the same methods.) The outer range ring, defining the outer boundary of the transition airspace, is modeled as a circle centered on a point near the airport; we can consider the circle to be, approximately, at the position where aircraft make the transition from level en route flight to their descent into the airport.

In the *Free Flight* approach, each aircraft is routed as it arrives, on its most favorable route, taking into account the constraints and the flights that have arrived before it. This can be implemented via a greedy algorithm that considers flights on a first come first serve basis, as is done in the simulation of Free Flight in the Flow-Based Route Planner (FBRP) [39, 40] system. FBRP greedily routes flights by searching space-time for optimal (shortest) routes for each aircraft in succession. This heuristic approach excels in its simplicity but the flow patterns that result are unstructured and thus have a high routing complexity (see [28] for the definition of a suitable “complexity” metric). Another possible approach is to establish disjoint “tubes” in space-time and then pack aircraft “head to tail” into each tube; the FBRP algorithm computes such trajectories of aircraft that are clustered into “flows” along a common route for a certain window of time (the duration of the flow). Usually the inner range ring becomes the bottleneck for either greedily established routes or globally planned tubes and this potentially decreases the capacity of the transition airspace. Our approach explicitly constructs merge trees using optimization, via a dynamic programming algorithm, and thereby avoids the bottleneck by planning the placement and connections among merge points of the arrival tree.

2 Preliminaries

In this chapter we briefly review some known results on thick paths and maximum flows on polygonal domains; see Mitchell [34] and Polishchuk [38] for complete treatments on these topics.

Continuous Max-flow Min-cut Let Ω be a compact and connected 2-dimensional domain, consisting of an outer (simple) polygonal boundary Γ and a set $\mathcal{H} = \{H_1, H_2, \ldots, H_h\}$ of polygonal holes. Let n be the total of vertices on the boundaries of Ω, including the outer boundary Γ and the inner boundaries (the holes $\mathcal{H}$). A flow in Ω is a divergence-free vector field σ (div $\sigma = 0$) with support in Ω. Flows enter Ω at Γ_s and exit at Γ_t. The value of any flow σ is the integral of σ across s or t, i.e.,

$$\int_{\Gamma_s} \sigma \cdot \boldsymbol{n}\, ds,$$

where $\boldsymbol{n}$ is the unit outward normal vector to Γ and s is the natural parameter of Γ. The objective is to find a convergence-free vector field with the maximum flow value.

An $s-t$ cut of Ω is the partition of Ω into two sets S and T, $P = S \cup T$, $S \cap T = \emptyset$, such that $\Gamma_s \in S, \Gamma_t \in T$. The capacity of the cut is the length of the boundary between S and T, measured according to the weighted 0/1 metric, which assigns cost 1 for traveling through P and cost 0 for traveling through its holes. The minimum cut, or mincut, through P is a cut of minimum capacity.

Strang [49] has proved the continous max-flow/min-cut theorem: the value of the maximum flow is equal to the capacity of a minimum cut. Mitchell [34] gave an $O(nh + n\log n)$ algorithm to find a maximum flow using a "continuous-Dijkstra" paradigm.

Another important concept is the *critical graph* [19, 34] associated with domain Ω, which is intimately related to the mincut. Let B (resp., T) be the portion of Γ counterclockwise between Γ_s and Γ_t (resp., between Γ_t and Γ_s); we sometimes refer to B as the "top" and T and the "bottom". We define a graph G that has a vertex for every hole in $\mathcal{H}$ and one vertex for T and one vertex for B, i.e, $G = (V, E)$ where $V = \mathcal{H} \cup \{T, B\}$. For any two holes H_i

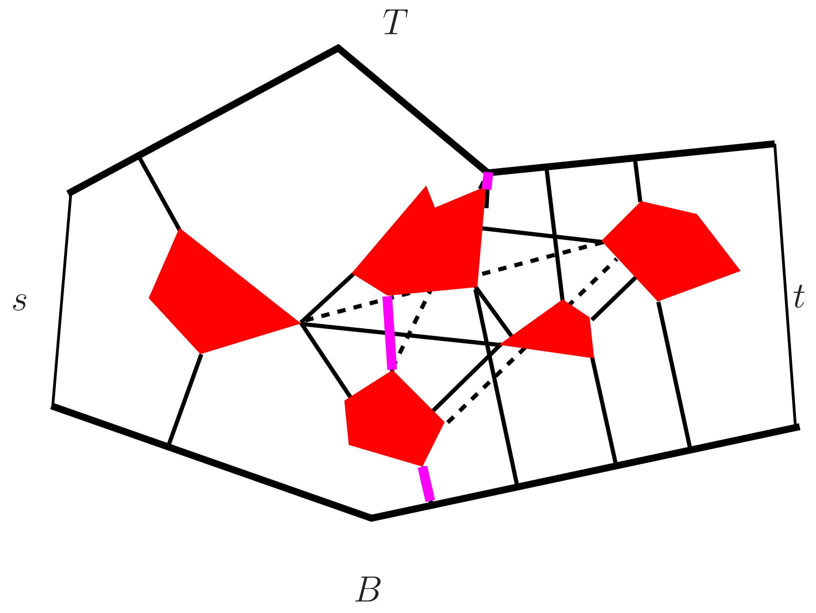


Figure 1: A critical graph of a polygonal domain with holes. The mincut is shown in magenta.

and H_j, there is an edge between their corresponding vertices, (i, j), whose weight is the minimum Euclidean distance between H_i and H_j; there is also an edge between any H_i and the "top" chain T (resp., B) whose weight is the minimum Euclidean distance between H_i and T (resp., B). (Essentially, it is sufficient to connect only those holes between which the shortest segment does not intersect other holes.) This weighted (combinatorial) graph G is called the *critical graph* associated with domain Ω; see Figure 1. Mitchell [34] showed that G must contain the mincut and in fact the mincut is (represented by) the shortest path between T and B in G.

A critical graph G can be constructed in $O(nh)$ time by using the linear time algorithm by Amato [2] to find the minimum distances (separations) between polygonal holes and between polygonal holes and T or B. The value of the maximum flow can then be determined by searching for the shortest $T - B$ path in G.

Discrete Continuous Max-flow Min-cut (Thick Paths) This result is an extension of the continuous max-flow/min-cut theorem and was studied in [38]; here "Discrete" refers to the fact that the flowlines are not allowed to be arbitrarily thin and must decompose into a finite set of *thick paths*. A

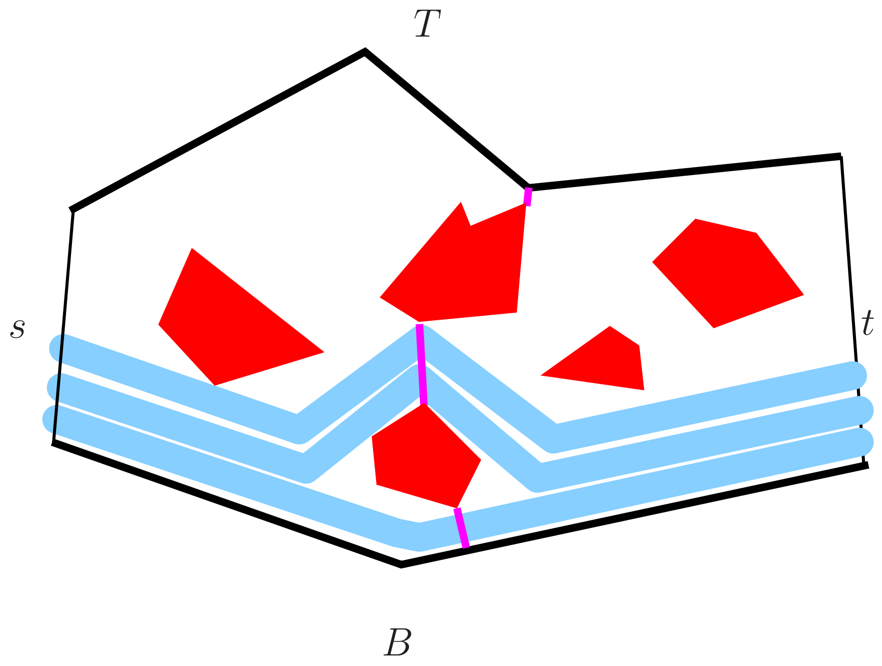


Figure 2: The maximum number of thick paths that correspond to the length of the mincut.

w-thick path, or *lane of width w*, is the Minkowski sum of a usual ("thin") source-to-sink path and a disk of radius $w/2$ centered at the origin (refer to [36] for more definitions and background). We consider the parameter w to be fixed and refer to a w-thick path simply as a thick path. Instead of finding a continuous vector field, the goal here is to compute the maximum number of pairwise-disjoint thick paths within Ω from Γ_s to Γ_t.

The *thresholded critical graph* [38], $G_{\lfloor\rfloor}$, of domain Ω is defined the same way as with the critical graph G except that the length of every edge is that of G divided by w and rounded down to the nearest integer. The discrete continuous max-flow/min-cut theorem [3, Thm. 2.3] states that the maximum number of thick non-crossing paths in Ω is equal to the shortest $T - B$ path length in $G_{\lfloor\rfloor}$; an exemplification is shown in Figure 2. Polishchuk [3, Thm. 2.2] also showed that the maximum number of thick non-crossing paths, together with a representation of the paths, can be found in $O(nh + n \log n)$ time by simulating wavefront propagation from the top (or the bottom); this algorithm is sometimes called the "uppermost paths" algorithm as it tries to route paths one by one as close to the current "top" as possible.

3 Approximating Maximum Flows

This chapter represents joint work with Joondong Kim and Joseph S. B. Mitchell, used with their permissions, and it originally appeared in a conference proceedings [23]. In this chapter we consider the discrete continuous max-flow problem described in chapter 2 and use the technique of geometric spanners to derive an approximation algorithm to compute the value of the maximum flow that runs in sub-quadratic time.

3.1 Introduction

We consider the problem of routing multiple disjoint "thick" paths through a polygonal domain in the plane. The problem arises in various application domains, including VLSI wiring, robotics, sensor networks, and ATM. Our motivation comes from an ATM application in which the goal is to compute the "capacity" of an airspace: find the maximum number of disjoint "air lanes" avoiding hazardous weather and other "constraints" (obstacles) within an airspace of interest (e.g., a "flow-constrained area") [29, 30]. The goal is to provide computer-automated decision support tools for the Next Generation Air Transportation System [50], specifically, to perform "capacity estimation" on an airspace to determine its maximum throughput, which measures how constrained the airspace is. We have further applied the capacity estimation algorithm for automated "flow constrained areas" (FCA) determination [55], which can then be used to facilitate strategic path planning for flights.

Problem Formulation The input to our problem is a polygonal domain Ω, consisting of an outer polygon Γ and a set $\mathcal{H} = \{H_1, H_2, \ldots, H_h\}$ of h holes. Let n denote the total number of vertices of Ω. In this chapter we focus on the special case in which $\mathcal{H}$ consists of a set of *point holes*; thus Γ is a simple polygon with $n - h$ vertices. This is the special case that arises in our ATM application, since the weather data is typically given as a set of points (pixels) at which there is hazardous weather predicted, with each point having an intensity value representing the severity of weather at that point; see Figure 3.

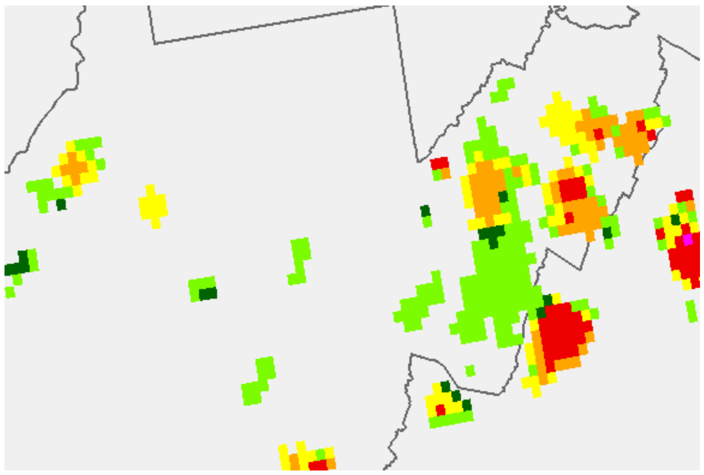


Figure 3: One snapshot of weather around West Virginia, with pixels shown as colored squares.

A *w-thick path*, or *lane of width w*, is the Minkowski sum of a usual ("thin") source-to-sink path and a disk of radius $w/2$ centered at the origin. We consider the parameter w to be fixed and refer to a w-thick path simply as a *thick path*. Two edges, Γ_s and Γ_t, of Γ are designated as the *source* and the *sink*; flows, or thick paths, enter Ω at Γ_s and exit at Γ_t. Our goal is to compute, approximately, the maximum number of pairwise-disjoint thick paths within Ω from Γ_s to Γ_t.

Geometric Spanners Given a set of n points $S = \{p_1, p_2, \ldots, p_n\}$ in the Euclidean plane, a *geometric spanner* is a spanning subgraph, G, of the complete graph on S such that the shortest path distance between any pair of points p_i and p_j in G, $d_G(p_i, p_j)$, approximates the Euclidean distance, $d(p_i, p_j)$, between the two points. If there exists a real number $t > 0$ such that for any pair of points p_i and p_j, $d_G(p_i, p_j) \leq t \cdot d(p_i, p_j)$, we say that G is a *t-spanner* for S. The maximum dilation for all pairs of points, $\max_{i,j} d_G(p_i, p_j)/d(p_i, p_j)$, is defined as the *stretch factor* of G with respect to S. It is obvious that, for a t-spanner, t is an upper bound on its stretch factor.

Analogous to the above definitions on geometric spanners, we can also define spanner for general weighted graphs, or networks, by replacing the Euclidean metric by the shortest path metric in the original graph, and we refer to this type of spanners as *graph spanners*. It is worth noting that the spanner results we will prove in Section 3.2 are actually in the sense of graph spanners, because we are approximating the rounded-down Euclidean metric, although the techniques we are using to derive the results are for geometric spanners.

It is known that a constant stretch factor geometric spanner exists [12], and that the Delaunay graph (Delaunay triangulation) on S is one such spanner with stretch factor known to be between 1.581 ($> \pi/2$) [6] and $\frac{4\pi}{3\sqrt{3}}$ (≈ 2.42) [22]. It remains an open problem what is the true stretch factor gap for Delaunay graph.

It is also known how to construct geometric spanners with stretch factors arbitrarily close to 1, which we refer to as $(1+\varepsilon)$-*spanners*. The Yao graph in [54] and the θ-graph in [22] are examples of $(1+\varepsilon)$-spanners. Specifically, $(1+\varepsilon)$-spanner is a family, $\mathcal{G}$, of geometric spanners for S such that for any real number $\varepsilon > 0$ there is a graph $G_\varepsilon \in \mathcal{G}$ that is a $(1+\varepsilon)$-spanner for S, can be constructed in $O(n \log n)$ time and is of size $O(n)$; note that both the time and size complexity bounds have a hidden constant that depends on $1/\varepsilon$.

Related Work Algorithms for computing maximum flows and minimum cuts in the continuum (2D geometric domains) were first studied by Mitchell [34]. Recent results have examined the problems of minimizing path lengths for multiple thick paths (minimum-cost flow) [36] and routing thick paths in dynamic environments (e.g., moving weather systems) [3]. Polishchuk's thesis [38] is a good reference for results in thick paths and maximum flows. The application of max-flow techniques in ATM is addressed, e.g., in [29, 30].

Related to our experiment where we study the stretch factor of Delaunay graph under the "rounded" Euclidean metric, [17, Sec. 6] studied the average and maximum stretch factor, among other spanner properties, of several well-known geometric spanners for random point sets.

Summary of Results Using the propagation algorithm of [34], appropriately modified to handle discrete thick paths (versus continuous flow fields), our optimization problem can be solved exactly in time $O(nh + n \log n)$ (see [3, Thm. 2.2]), for a polygonal domain with h polygonal holes, having a total of n vertices. In this work we propose a simple 1/2-approximation algorithm for the case that Ω is a polygonal domain with point holes. Our algorithm searches a Euclidean spanner graph for an approximate min-cut, in time $O(n + h \log(nh))$. We show that this results in an approximation for the problem of maximizing the number of disjoint thick paths.

Experimentally, we use the Delaunay diagram as the Euclidean spanner and we show that the stretch factor of our search graph constructed using the Delaunay diagram tends to be bounded by 2 in practice, on both real weather data and randomly generated data; this suggests that Delaunay diagram could also be a good candidate for spanner to get 1/2-approximation. We investigate how the stretch factor changes as the number of points in the random data increases, and also as the average density of the points changes.

3.2 An Approximation Bound

We consider the *thresholded critical graph* G of Ω as described in chapter 2, i.e., $G = (V, E)$ where $V = \mathcal{H} \cup \{T, B\}$, $E = \{(i, j) | i, j \in V, i \neq j\}$. The weight of edge (i, j) is $c(i, j) = \lfloor d(i, j)/w \rfloor$, where $d(i, j)$ is the minimum Euclidean distance between the corresponding holes (or between a hole and a "top" chain T or a "bottom" chain B), and w is the path width. Let $G_{\mathcal{H}}$ be the subgraph of G induced by vertices in $\mathcal{H}$. Finally, we select any geometric $(1+\varepsilon)$-spanner and let $G^{\varepsilon}_{\mathcal{H}}$ denote the subgraph of $G_{\mathcal{H}}$ whose edge set consists of the union of the edges of the spanner and the set of Delaunay edges of $\mathcal{H}$, with edge weights remain as in G (the rounded-down distance). We will first show that $G^{\varepsilon}_{\mathcal{H}}$ is a 2-spanner for $G_{\mathcal{H}}$ in Lemma 3.1. By augmenting $G^{\varepsilon}_{\mathcal{H}}$ to a new graph G^{ε} by adding connections from each point hole H_i to both the top and the bottom, G^{ε} becomes a 2-spanner for the thresholded critical graph G while maintaining $O(h)$ in size. We then just need to search for the shortest $T - B$ path in G^{ε} to get an approximation to the maximum flow value.

Lemma 3.1. *For $\varepsilon \leq 0.5$, $G^{\varepsilon}_{\mathcal{H}}$ is a 2-spanner for $G_{\mathcal{H}}$.*

Proof. It suffices to show that for any edge (u, v) in $G_{\mathcal{H}}$, there is a path from u to v in $G^{\varepsilon}_{\mathcal{H}}$ of length at most $2 \cdot c(u, v)$. There are two cases to consider,

Case 1: $c(u, v) = \lfloor d(u, v)/w \rfloor = 0$. Since $G^{\varepsilon}_{\mathcal{H}}$ includes Delaunay edges, we know from [7, 12] that there exists a path π_1 in $G^{\varepsilon}_{\mathcal{H}}$ with vertices $u = v_1, v_2, \ldots, v_k = v$ such that $\forall i, d(v_i, v_{i+1}) \leq d(u, v)$. Since $c(u, v) = 0$ implies that $d(u, v) < w$, we have that $\forall i, c(v_i, v_{i+1}) = 0$, so the path π_1 has zero length in $G^{\varepsilon}_{\mathcal{H}}$ and the claimed stretch factor automatically holds.

Case 2: $c(u, v) > 0$. Since $G^{\varepsilon}_{\mathcal{H}}$ contains an Euclidean $(1+\varepsilon)$-spanner, there exists a path π_2 in $G^{\varepsilon}_{\mathcal{H}}$ such that $\sum_{i=1}^{k-1} d(v_i, v_{i+1}) \leq (1 + \varepsilon) d(u, v)$. Thus, the stretch factor of $G^{\varepsilon}_{\mathcal{H}}$ is at most

$$\begin{aligned}\frac{\sum_{i=1}^{k-1} c(v_i, v_{i+1})}{c(u,v)} &= \frac{\sum_{i=1}^{k-1} \lfloor d(v_i, v_{i+1})/w \rfloor}{\lfloor d(u,v)/w \rfloor} \\ &\leq \frac{\lfloor \sum_{i=1}^{k-1} d(v_i, v_{i+1})/w \rfloor}{\lfloor d(u,v)/w \rfloor} \\ &\leq \frac{\lfloor (1+\varepsilon) d(u,v)/w \rfloor}{\lfloor d(u,v)/w \rfloor}.\end{aligned}$$

We show in the following proposition (Proposition 3.2) that if $f(x) = \frac{\lfloor (1+\varepsilon)x \rfloor}{\lfloor x \rfloor}$ and $\varepsilon \leq 0.5$, then $f(x) \leq 2$, for $x > 0$. □

Proposition 3.2. *The function $f(x) = \frac{\lfloor (1+\varepsilon)x \rfloor}{\lfloor x \rfloor}$, $x > 0$ is bounded above by 2 when $\varepsilon \leq 0.5$.*

Proof. Note that $1 + \varepsilon \leq 3/2$ and that $f(x)$ is a step function whose value changes only when $x = n$ or $x = \frac{n}{1+\varepsilon}$, for $n \in \mathbb{Z}^+$.

When $x = n$, $n \in \mathbb{Z}^+$, $f(x) \leq \frac{(1+\varepsilon)n}{n} = 1 + \varepsilon \leq 3/2$.

When $x = \frac{n}{1+\varepsilon}$, $n \in \mathbb{Z}^+$, $f(x) = \frac{\lfloor n \rfloor}{\lfloor n/(1+\varepsilon) \rfloor} \leq \frac{\lfloor n \rfloor}{\lfloor 2n/3 \rfloor}$. Let $g(n) = \frac{\lfloor n \rfloor}{\lfloor 2n/3 \rfloor}$. If $n = 3k$, $g(n) = \frac{3}{2}$. If $n = 3k-1$, $g(n) = \frac{3k-1}{2k-1}$ and it achieves its maximum of 2 at $k = 1$. If $n = 3k+1$, $g(n) = \frac{3k+1}{2k}$ which also achieves its maximum of 2 at $k = 1$. Thus $f(x) \leq 2$ in this case. □

With the above lemma in place, we are able to derive the following theorem,

Theorem 3.3. *The maximum number of disjoint w-thick paths in a polygonal domain with n vertices and h point holes can be $\frac{1}{2}$-approximated in time $O(n + h \log(nh))$*

Proof. $G^{\varepsilon}_{\mathcal{H}}$ is constructed in time $O(h \log h)$, the time needed to build the Delaunay graph or a $(1+\varepsilon)$-spanner (with $\varepsilon = 0.5$ and $O(h)$ edges) for h points. We then construct a graph G^{ε} from $G^{\varepsilon}_{\mathcal{H}}$ by adding nodes for B and T, and linking these nodes to each point of $\mathcal{H}$. In order to compute the distances from each point of $\mathcal{H}$ to the polygonal chains T and B, we spend time $O(n)$ to construct the Voronoi diagram of T (resp. B) [10] with sites being open segments and vertices from T (resp. B), and preprocess the Voronoi diagram into a point location data structure. Since the Voronoi bisectors are algebraic curves with degrees at most 2, we can partition the Voronoi diagram into a monotone subdivision using at most $O(n)$ vertical tangents, and then build the point location data structure in $O(n)$ time [13], allowing us the compute the distances from each point of $\mathcal{H}$ to B and to T in time $O(\log n)$ per point of $\mathcal{H}$. This augmented graph G^{ε} has $O(h)$ edges and is a 2-spanner for the critical graph G.

By the continuous max-flow/min-cut theorem [3, 34, 34], we know that the maximum number, OPT, of thick paths from source to sink is equal to the length, $|\pi_G|$, of a shortest path from B to T in G. Since G^{ε} is a 2-spanner for G, we know that the length, $|\pi_{G^{\varepsilon}}|$, of a shortest B-to-T path in G^{ε} is at most $2|\pi_G|$: $OPT \leq |\pi_{G^{\varepsilon}}| \leq 2 \cdot OPT$, i.e., $(1/2) \cdot OPT \leq (1/2)|\pi_{G^{\varepsilon}}| \leq OPT$. Thus, $(1/2)|\pi_{G^{\varepsilon}}|$ is a $\frac{1}{2}$-approximation to the maximum number of source-to-sink thick paths.

Our algorithm takes time $O(h \log h)$ to build $G^{\varepsilon}_{\mathcal{H}}$, $O(n+h \log n)$ to build G^{ε} from $G^{\varepsilon}_{\mathcal{H}}$, and another $O(h \log h)$ to search for a shortest path in G^{ε}. Adding things up, the total running time is $O(n + h \log(nh))$. □

3.2.1 Using Delaunay Graph

If instead of using a $(1+\varepsilon)$-spanner with $\varepsilon \leq 0.5$ we use the Delaunay graph as a 2.42-spanner, the resulting spanner graphs $G^{\varepsilon}_{\mathcal{H}}$ and G^{ε} become 4-spanners, which we denote by $G^{Del}_{\mathcal{H}}$ and G^{Del}, respectively. (In fact, $G^{Del}_{\mathcal{H}}$ is just the Delaunay graph on $\mathcal{H}$ endowed with rounded-down edge weights.)

Corollary 3.4. *Using the Delaunay graph as a 2.42-spanner, $G^{Del}_{\mathcal{H}}$ is a 4-spanner for $G_{\mathcal{H}}$ and G^{Del} is a 4-spanner for G.*

Proof. Everything follows directly from the proof of Lemma 3.1, only with $\varepsilon \leq 0.5$ replaced by $1 + \varepsilon = 2.42$. Because the function $f(x) = \frac{\lfloor 2.42x \rfloor}{\lfloor x \rfloor}$, $x > 0$ retains its maximum value 4 at $x = 4/2.42$, $G^{Del}_{\mathcal{H}}$ is a 4-spanner for $G_{\mathcal{H}}$ and thus G^{Del} is a 4-spanner for G. □

We now know that $G^{Del}_{\mathcal{H}}$'s stretch factor is bounded by 4, and we also have the following example, Figure 4, showing that it can be at bad as 2. In this figure, the Delaunay graph for the eight points is structured as two copies of the same diamond (rhombus) connected by three parallel chords; the longer diagonal in the diamond measures $\sqrt{3}$. The shortest path between s and t is 2 in $G_{\mathcal{H}}$ by taking the blue dotted path, and it is 4 in $G^{Del}_{\mathcal{H}}$ by going around the diamonds. This clearly shows the stretch factor of $G^{Del}_{\mathcal{H}}$ in Figure 4 is 2.

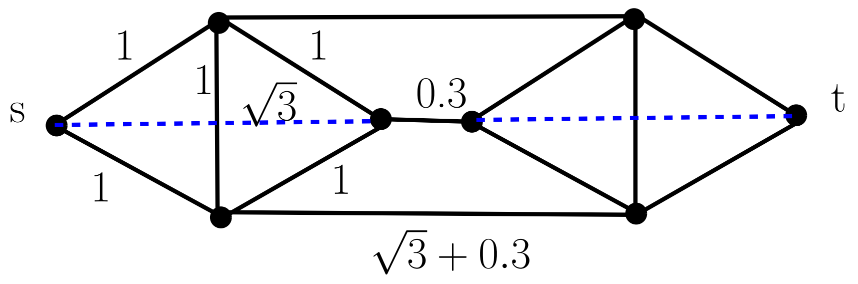


Figure 4: Example showing $G_{\mathcal{H}}^{Del}$ can have stretch factor 2. Bold lines plot the Delaunay graph and the numbers are the original Euclidean distances. The shortest path distance between s and t is 2 (blue dotted lines) in $G_{\mathcal{H}}$, and 4 in $G_{\mathcal{H}}^{Del}$.

3.3 Experiments

We did experiments based on computing a specific spanner, G^{Del}, using the Delaunay graph of the points $\mathcal{H}$. Theorem 3.3 tells us that if we use a spanner with stretch factor bounded by 1.5 ($\varepsilon = 0.5$), then our approach gives a $\frac{1}{2}$-approximation. Since the Delaunay graph is only known to be a $\frac{4\pi}{3\sqrt{3}} \approx 2.42$-spanner (as of the date of writing), we only have a theoretical guarantee from Corollary 3.4 that G^{Del} is a 4-spanner and thus the Delaunay-based approach gives a $\frac{1}{4}$-approximation. However, we will see that, in practice, the Delaunay performs very well and the stretch factors of G^{Del} for the point samples we generated is never more than 2.

In our experiments, we use VRONI [20] to compute the Delaunay graph of the point set $\mathcal{H}$, augment it to a spanner of the thresholded critical graph as described in Theorem 3.3, compute its shortest $T - B$ path length, calculates the ratio of it over the actual length of the mincut and record this value as the stretch factor of our search graph G^{del} constructed using Delaunay graph. (Although this is not the true stretch factor of G^{del} as we are ignoring the stretch factors for other pair of vertices, this is the only stretch factor that matters to the approximation ratio in Theorem 3.3.) We use a unit square box as the outer boundary of the polygonal domain Ω, and use two types of input data for the point holes $\mathcal{H}$: (1) uniformly generated points, and (2) real weather data, scaled to the unit square. For both sets of input data, we examine the relationship between the stretch factor as a function of the "average density" of the point set $\mathcal{H}$. We use the average edge length in the nearest neighbor graph of $\mathcal{H}$ to measure the *average density* of the points $\mathcal{H}$; the smaller this *average nearest neighbor distance* is, the denser the point set is.

3.3.1 Random Point Sets

In the experiments with random point sets, we vary the number, h, of points and the lane width, w.

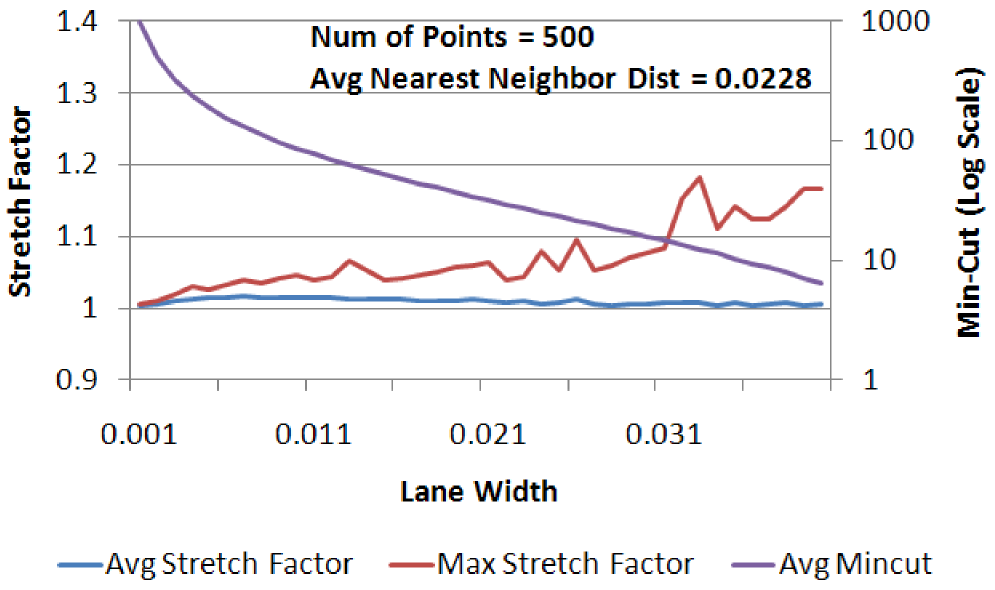


Figure 5: Average and maximum stretch factor and length of min-cut as a function of lane width w for random points.

Varying Lane Width For the experiments with varying lane width w, we fix the number, $h = 500$, of random points and vary w from 0 to 0.4. As the total number of points is fixed at 500, the average nearest neighbor distance is always around 0.0228 for all instances. For each w, we generated 100 random instances and compiled simple statistics – the average and the maximum stretch factor. Figure 5 shows that the average stretch factor is usually very close to 1. For maximum stretch factor, it tends to go up as the lane width w increases. Intuitively, when w is much smaller than the average nearest neighbor distance, the rounded-down Euclidean metric behaves more or less like the original Euclidean metric and stretch factors of G^{del} tend to be good; it is only when w is of comparable scale with the average nearest neighbor distance that the rounding-down starts to take effect (and it could be more likely that there are points of distance slightly less than $2w$ which gets rounded down to 1 while the approximate path between them in the spanner has two edges of length slightly bigger than w thus creating a stretch factor of 2 between the two points).

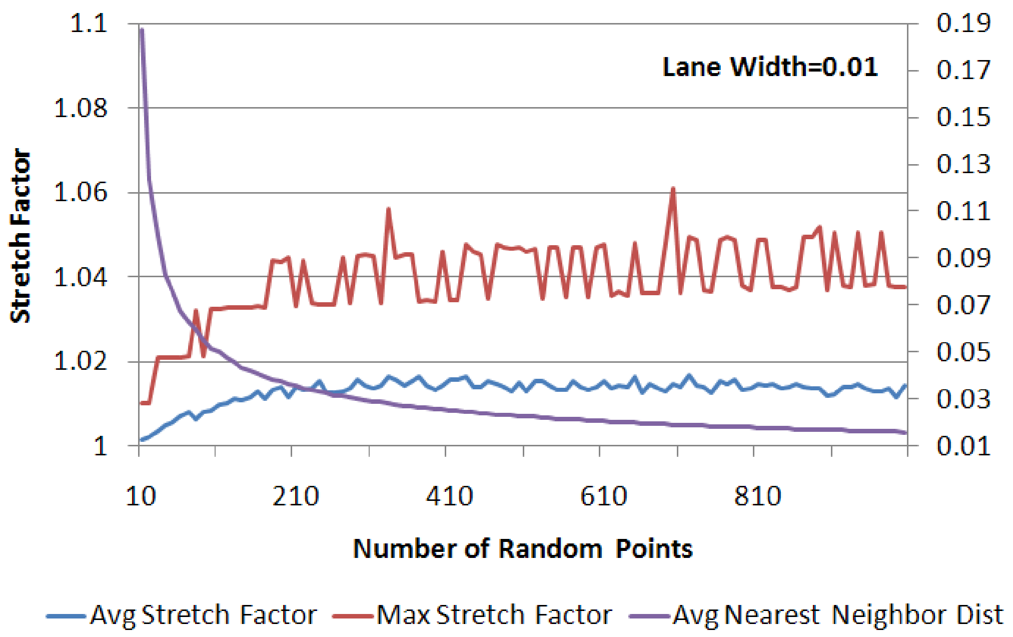


Figure 6: Average and maximum stretch factor and the average nearest neighbor distance as a function of the number of random points.

Varying the Number of Points For the experiments with varying number h of points, we fix the width, $w = 0.01$, and vary h from 0 to 1000 in increments of 10. For each h, we generated 100 random instances and recorded the average and maximum stretch factors, and we have also computed the average nearest neighbor distance for the 100 instances. We plot the average and maximum stretch factor and the average nearest neighbor distance against the number of random points in Figure 6. We also plot, for the same data, the average and maximum stretch factor against the average nearest neighbor distance in Figure 7.

The results show that both the maximum and the average stretch factor is close 1 in all cases. Figure 6 shows that the maximum stretch factor goes up initially as the number of points increases and the average nearest neighbor

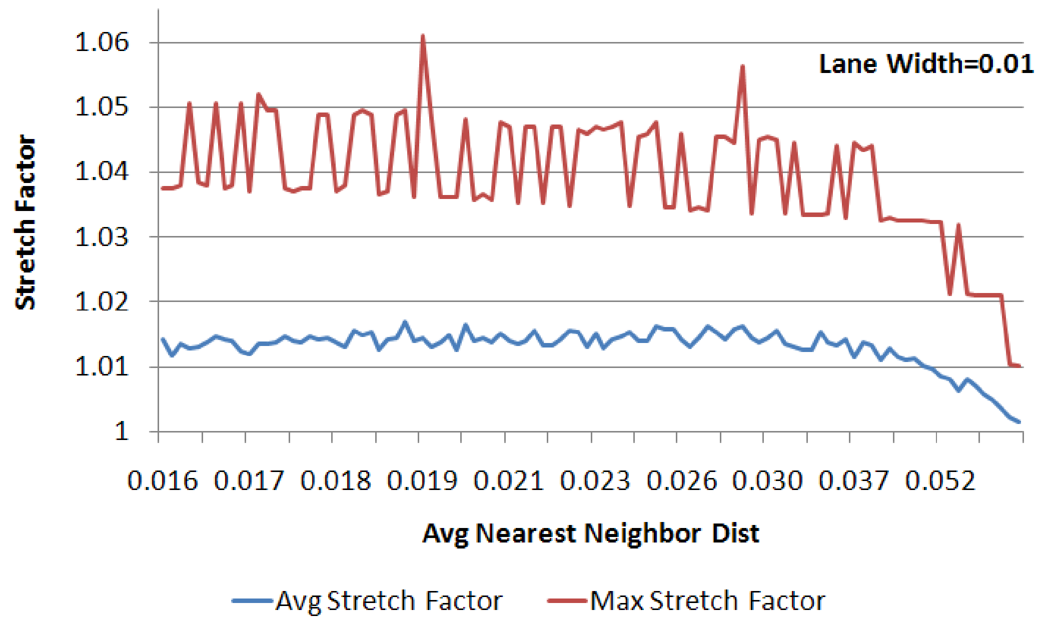


Figure 7: Average and maximum stretch factor as a function of the average nearest neighbor distance.

distance decreases accordingly; this is because when the average nearest neighbor distance is much smaller than w, most of the edge weights get rounded down to 0 and thus the shortest $T-B$ path in both G^{Del} and G are very likely to be the same. Only when the average nearest neighbor distance becomes comparable to w does the maximum stretch factor start to increase; and it stays almost put when the average nearest neighbor distance drops to and stays close to 0.02. The above observations are further confirmed in Figure 7 when we plot the stretch factors directly against the average nearest neighbor distance.

3.3.2 Real Weather Data

Table 1 shows the stretch factor data for real weather data. Since real weather tends to have clusters of weather points (pixels), the average nearest neighbor distance is much smaller than for random point sets; accordingly, we set the lane width to be very small, $w = 0.005$. (For $w = 0.05$, we found that the stretch factor is always 1.) The results show that the stretch factor is very close to 1, even if the average nearest neighbor distance is comparable to w.

	h	min-cut	avg nearest neighbor distance	stretch factor
Set1	576	179	0.0039	1.0056
Set2	502	182	0.0039	1.0110
Set3	430	184	0.0048	1.0163
Set4	752	177	0.0038	1.0113
Set5	820	180	0.0028	1.0000

Table 1: Results for real weather data with $w = 0.005$. Here h is the number of point obstacles.

3.4 Conclusion

Our goal has been to explore the use of spanners in computing approximations for maximizing the number of pairwise disjoint thick paths that can be routed through a polygonal domain in the plane. The advantage of using a spanner (e.g., Delaunay graph) for computing minimum cut values approximately is that it gives a linear space and near-linear time simple algorithm in place of the far more complex exact $O(nh + n \log n))$ algorithm, or the naive $O(n^2)$ algorithm that is easiest to implement. We have seen experimentally that the Delaunay graph does very well in most cases; thus, the Delaunay-based approximation is likely an effective and practical means of doing capacity estimation for ATM.

The exact min-cut problem is a shortest path problem in the plane that may be solvable in $O(n \log n)$ exactly, e.g., by a variant of the continuous Dijkstra paradigm that solves obstacle-avoiding shortest paths in $O(n \log n)$ time. Also, we are examining possible improved approximations possible using spanner techniques, and we are now developing algorithms to produce a set of disjoint thick paths that achieve the capacity determined by our approximation algorithms. Finally, we are doing further experimentation with other Euclidean spanner graphs, with stretch factors approaching 1.

4 Thick Paths with Different Widths

This chapter represents joint work with Joondong Kim, Joseph S. B. Mitchell, Valentin Polishchuk, and Shang Yang, used with their permissions, and it originally appeared in part in a technical report [24]. In this chapter we consider a variant of the discrete continuous max-flow problem, or the thick paths problem, in which thick paths can be of different widths. Specifically, for a given polygonal domain Ω with a set of holes $\mathcal{H} = \{H_1, H_2, \ldots, H_h\}$ and K different path widths (or lane widths), $w_1, w_2, \ldots, w_K$, we want to determine if it is possible to route, for each k, N_k width-w_k paths from source to sink such that each path avoids the holes $\mathcal{H}$ and that no two paths overlap. We show that this problem is NP-hard, even when there are only two distinct path widths, $w_1 = 2$ and $w_2 = 3$. We also give a Dijkstra-type algorithm to solve the problem in polynomial time when a "width sequence" along the source is given.

4.1 Introduction

Motivation The problem we are considering here arises from the need to do capacity estimation for aircraft of different performance capabilities and thus with different Required Navigation Performances (RNPs). In our model, RNP and *required lateral separation distance*, δ, together determine the lane width; a RNP-n lane corresponds to a lane width of $4n + \delta$ where δ is usually a constant. An aircraft can utilize an air lane only if its RNP is at or below the RNP associated with that lane; an aircraft of low performance capability, meaning large RNP, is not permitted on an air lane of small lane width. Good capacity estimation and route planning algorithms must take the difference in RNPs into account by possibly fitting aircraft of small RNPs into smaller gaps in the airspace, which would not be pass-able by aircraft of large RNPs. See Figure 8.

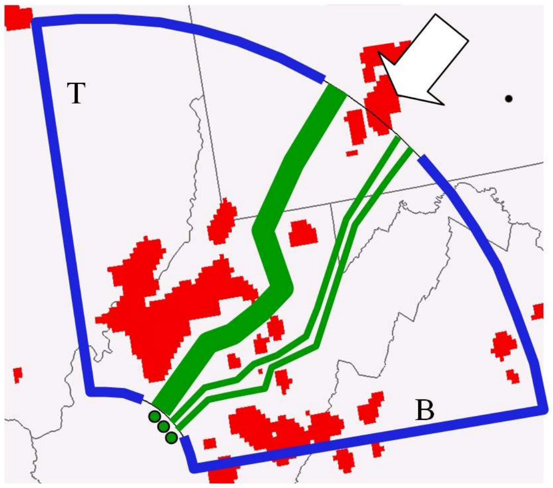


Figure 8: Within a quadrant, some width permutation sequences yield capacity for all widths values, some do not.

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 [9]. 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 in [51]. 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 to roadmap-based to decoupled (see [51] for details). A polynomial-time algorithm for finding a maximum number of thick paths (of the same width) in a polygonal domain is presented in [3].

Problem Formulation The input to our problem is a polygonal domain Ω consisting of an outer polygonal boundary and a set of holes $\mathcal{H} = \{H_1, \ldots, H_h\}$. 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. The paths sought are of one of K widths, $w_1, \ldots, w_K$. Our goal is to determine, given a set of numbers $N_1, N_2, \ldots, N_K$, if it is possible to route N_k width-w_k paths from source to sink, so that each path avoids the holes $\mathcal{H}$ and that no two paths overlap.

The source and sink edges split the boundary of the outer polygon of the domain into two parts — the "top" T and the "bottom" B. We will assume 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 $\tau = (w_1, \ldots, w_M)$, $w_m \in \{1, \ldots, K\}$ be a sequence of path widths. We say that τ is the *width sequence* of a collection of M paths if the m-th path in the collection is of width w_m.

4.2 Hardness Result

For the mixed-width paths problem, we consider the special case that $K = 2$. Specifically, we consider only two kinds of widths, 2 and 3, and the decision question is whether there exist c_2 2-thick paths and c_3 3-thick paths. We will prove that the problem is NP-hard in the following theorem:

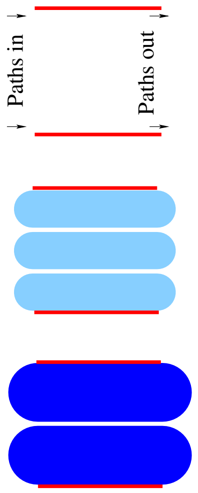
Theorem 4.1. *Given two numbers w_1 and w_2 and two integers c_{w_1} and c_{w_2}, it is NP-hard to decide whether it is possible to route c_{w_1} w_1-thick paths and c_{w_2} w_2-thick paths.*

The proof idea is based on the hardness proof idea used in [24] . We reduce from the INDEPENDENT SET. 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? Starting from G, we construct an instance of the mixed-width paths problem as follows.

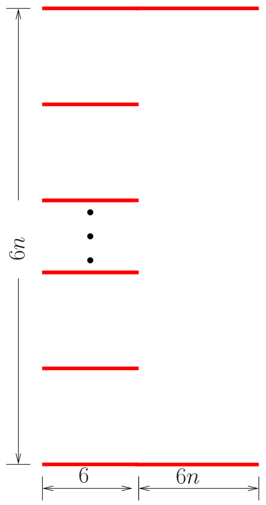
For each vertex of G we create a *vertex gadget* (Fig. 9(a)). We stack the n vertex gadgets one on top of another, forming the *vertex part* of the construction (Fig. 9(b), left part).

For each edge of G we create an *edge gadget*. To build the gadget, we first create an $6n$-by-$6n$ square with top and bottom sides being obstacles (Fig. 9(b), right part). If edge e is incident to a vertex i, we put a pair of point obstacles (a "gate") with distance 2 apart (thus allowing 2-thick path to pass but blocking 3-thick path) along the right side of the square. Finally, the top and the bottom boundary of each edge gadget are shifted by 2 from both up and down (Fig. 9(c)).

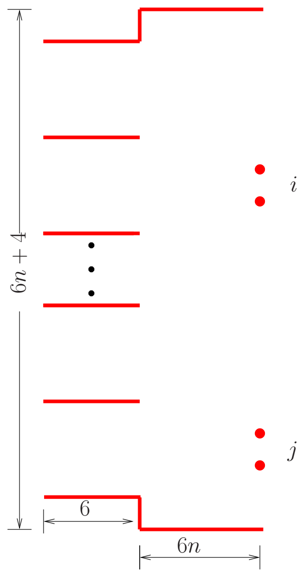
We finish the whole construction by putting the vertex part of the m edge gadgets side by side from left to right; we align the center of the "gate" with the center of the corresponding vertex gadget (Figure 9(c)). We claim that there exists an independent set of size k in G if and only if $3(n-k)$ 2-thick and $2k$ 3-thick paths can be routed all the way from the left of the construction to the right.



(a) Top: The vertex gadget is a 6-by-6 square with top and bottom sides being obstacles. Middle and bottom: if the paths, going through the gadget are of the same width, they are either (at most) three 2-thick path or (at most) two 3-thick paths.



(b) n vertex gadgets stacked, with an edge gadget connected to it from right. Each edge gadget is built from an $6n$-by-$6n$ square with top and bottom sides being obstacles.



(c) In the gadget for an edge (i, j), we put a pair of point obstacles with distance 2 apart in front of both vertex gadget i and j; also the top and the bottom sides of each edge gadget are shifted by 2 up and down.

Figure 9: Vertex and edge gadgets for the mixed-width paths problem

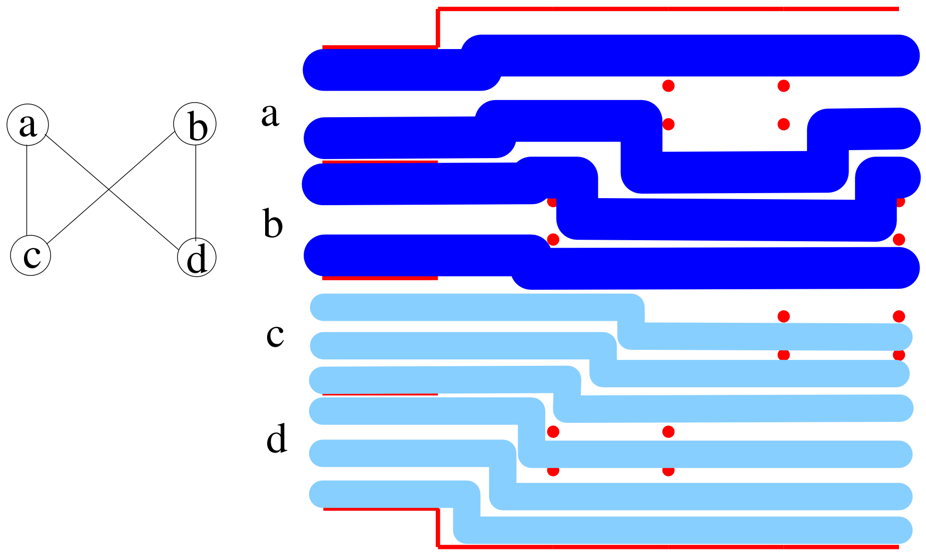


Figure 10: This example shows the construction for the graph on the left. The edge gadgets are not to scale. If there exists an independent set of size k in graph G, then we can route $3(n-k)$ 2-thick and $2k$ 3-thick paths.

Lemma 4.2. *If, for an arbitrary k, there exists $3(n-k)$ 2-thick paths and $2k$ 3-thick paths, then no vertex gadget has both 2-thick and 3-thick paths going through.*

Proof. Trivial from the construction of the gadgets; see Figure 9. □

Now suppose that G has an independent set of size k. We route two 3-thick paths through each of the vertex gadgets corresponding to vertices in the independent set. Route three 2-thick paths through each of the other vertex gadgets. We claim these paths may pass through all edge gadgets. Consider one edge gadget. If the gadget does not contain any "gate", the paths could pass through in the same as they come out from the vertex gadgets. If the gadget does contain a pair of "gates" corresponding to an edge (b, d) (refer to Fig. 10), at most one of the b, d vertex gadgets has two 3-thick paths going through it. WLOG, assume that vertex gadget b is above gadget d, and only gadget b is filled with two 3-thick paths. Since no "gate" can accommodate 3-thick path, we split the two 3-thick paths and route them above and below, "gate" b, respectively. The one above is shifted up by 1, causing all paths above it shifted by 1. The one below is shifted down by 1, causing paths below it but above those from vertex gadget d shifted down by 1 (refer to Fig. 10). For the

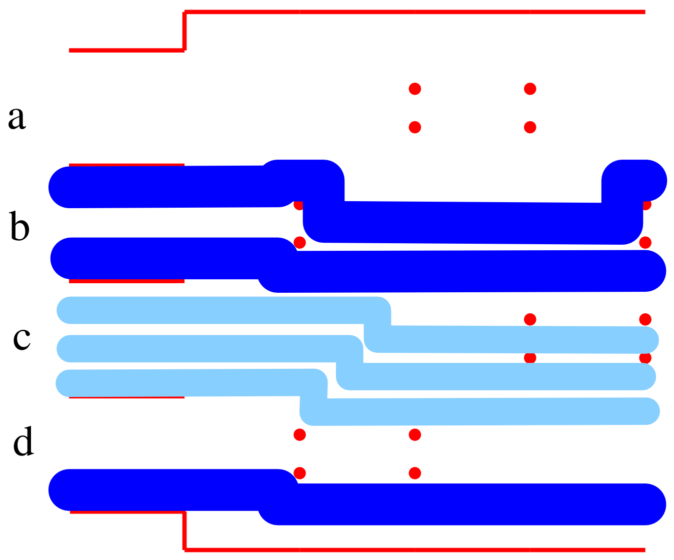


Figure 11: If there does not exist an independent set of size k in graph G, then we can not find $3(n-k)$ 2-thick and $2k$ 3-thick paths.

three 2-thick paths from gadget d, the top one can no longer be routed above "gate" d because there is only a height 1 channel there because of the shift. It has to go through "gate" d, and subsequently all paths below it are shifted down by 2. But because of the extra space of height 2 on both the top and the bottom of this edge gadget, the shifted paths can all be accommodated.

Then we suppose that there exist $3(n-k)$ 2-thick and $2k$ 3-thick paths. By Lemma 4.2, there are $n-k$ gadgets filled with 2-thick paths and k gadgets filled with 3-thick paths. Suppose that two gadgets, corresponding to endpoints of an edge (b, d) are both filled with two 3-thick paths (refer to Fig. 11). We claim that the $3(n-k)$ 2-thick and $2k$ 3-thick paths would not fit into the edge gadget (b, d). The two 3-thick paths from gadget b cannot be both routed above "gate" i because that would cause a up-shift of 4 of all paths above which is beyond the extra space of height 2 on the top. Thus one of them can be routed above and the other has to be routed below the "gate" which causes a down-shift of 1 towards "gate" d. Neither of the two 3-thick paths from gadget d can be routed above "gate" d because the previous down-shift. And they can not both be routed below as well that would cause another down-shift of 4 (refer to Fig. 11). This shows for any two vertex gadgets b, d corresponding to an edge, at most one of them can be filled with two 3-thick paths and thus all the vertex gadgets that batches of two 3-thick paths go through correspond to an independent set of G of size k. This establishes Theorem 4.1.

4.3 Given Width Sequence

Theorem 4.1 shows that the problem does not admit any polynomial time algorithm unless P=NP. However, if a width sequence τ is given, we can apply a variant of the "uppermost paths" algorithm, which we will describe in the next paragraph, used by Mitchell [34] and Polishchuk [38] to solve it optimally. When the number of path widths, K, and the N_i's are small, it is readily tractable to apply our methods to (possibly) every ordering and thereby determine exactly the solution to our problem.

Suppose that we are given a width sequence $\tau = (w_1, w_2, \ldots, w_M)$. *Uppermost* paths are thick paths routed "as close as possible" to the top T. Specifically, imagine that the free space of Ω is grass, over which fire travels at speed 1. Image also that the holes in Ω are composed of highly flammable material so that as soon as the fire reaches a hole, the entire hole is ignited instantaneously. We ignite T at time 0, and let Ω' be the part of the domain burnt by time w_1. The first *uppermost* path is a w_1-thick path routed within Ω' (refer to [38, Thm. 6.19], where the details of simulating the fire — handling events, attaching Riemann flaps, etc. — are given). The second uppermost path is the thick path routed iteratively by treating the lower boundary of the already routed w_1-thick path as the new T; the other uppermost paths are defined recursively in a similar manner. In [38] it was shown how to find the uppermost path in polynomial time by simulating wave (fire) propagation from T.

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 [36], 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.

The correctness of the above uppermost path algorithm is guaranteed by the following proposition:

Theorem 4.3. *If there exist a collection of thick paths in the domain Ω with width sequence τ, then there exist a collection of τ-uppermost paths. (A representation of) the τ-uppermost can be found in time polynomial in Ω, by* *propagating a wave from the top of the outer boundary polygon of* Ω.

Proof. This proof is identical to [38, Thm. 6.19]. Let $(\Pi_1, \Pi_2, \dots, \Pi_M)$ be a sequence of paths in Ω with width sequence $\tau = (w_1, w_2, \dots, w_M)$, i.e., each Π_i is a w_i-thick path. Start the fire propagation from T. Let Ω' be the part of Ω burnt by time w_1. Clearly Π_2 does not intersect Ω' because otherwise Π_1 would not fit within Ω'. So we route a uppermost w_1-thick path within Ω', and there exist a sequence of paths $(\Pi_2, \dots, \Pi_M)$ with width sequence $(w_2, \dots, w_M)$ within domain $\Omega \setminus \Omega'$. The proof follows by induction on the total number of the thick paths. □

We also have a "query" version of Theorem 4.3. In contrast with Theorem 4.3, the algorithm for Theorem 4.4 does not produce the paths; it just tests whether the routing is possible.

Theorem 4.4. *Given a type sequence* τ*, a graph can be built in time* $O(nh)$ *such that , 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.*

Proof. The *critical graph* of the domain [19, 34, 3] 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. 12). The graph can be built in $O(nh)$ time by using the linear time algorithm in [2] 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 [34]. 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 [19, 34, 3] 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. We keep the following invariant: The label $\ell(v)$ assigned to a vertex v is the largest index such that the paths $1, \dots, \ell(v)$ can be routed between T and the hole corresponding to v.

We start with assigning labels $\ell(T) = 0$ and $\ell(\cdot) = \infty$ for the other vertices. 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:

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

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.

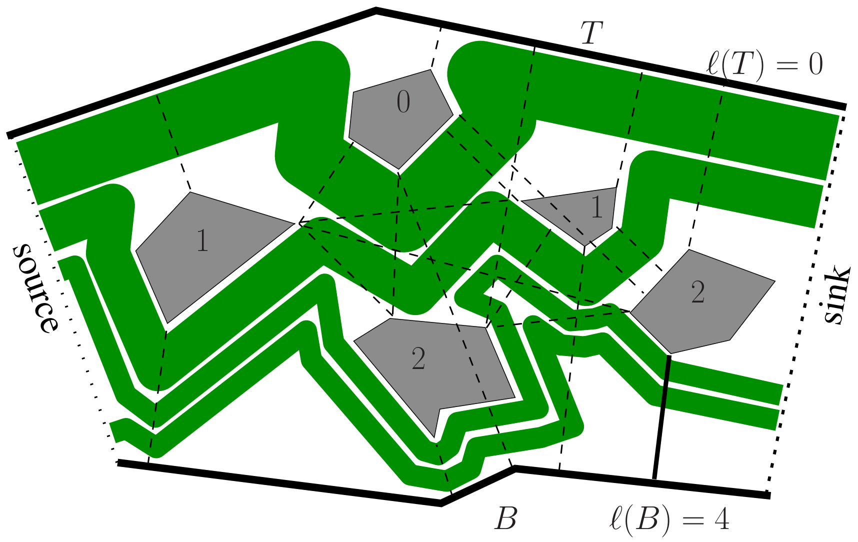


Figure 12: The edges of the critical graph are dashed. The numbers are labels of holes, corresponding to a sequence of widths $\tau = (3,2,1,1)$. For another sequence, say, $\tau' = (3,2,4)$, the label of B would have been 2 since there would not be room for the third path.

Just like in Dijkstra's algorithm, the induction on the label shows that the invariant holds after each step of the algorithm. In particular, 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.

Similar to the run-time analysis of Dijkstra's algorithm (using Fibonacci heap), our algorithm stated above takes time $O(M)$ to compute the partial sums of τ, $O(h^2 \log M)$ to relax all the edges in the critical graph and $O(h \log h)$ time to find the smallest labels in all rounds; these together sum up to $O(M + h^2 \log M)$.

□

4.4 Conclusion

We considered the problem of routing paths of multiple widths in a polygonal domain, which is motivated by capacity estimation for aircraft with different RNPs. The very basic decision version of the problem turns out to be NP-hard. We presented efficient algorithms for the special case when a width sequence is given.

We have left open the problem whether constant factor approximation exists for optimization versions of our problem, e.g., finding the maximum number of width-w_1 thick paths while guaranteeing at least N width-w_2 paths. For our hardness result on the two widths, 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 $w_1 = 1, w_2 = 2$. For our Dijkstra-like algorithm to test the feasibility of a given type sequence, we are also investigating if we can construct the paths in the same framework (without invoking the "uppermost path" algorithm).

5 Directional Capacity Estimation

This chapter represents joint work with Joondong Kim, Joseph W. Krozel, Jimmy Krozel, and Joseph S. B. Mitchell, used with their permissions, and it originally appeared in a conference proceedings [55]. In this chapter we study two methods of computing directional capacity for en route airspaces with hazardous weather constraints. Within a constant flight level, we analyze the dominant demand direction and then study two methods of measuring the directional capacity of the airspace. The first method, a grid-based method, considers eight fundamental directions for directional capacity, according to the standard directions of travel. The second method considers an arbitrary direction of flow and determines the directional capacity in that particular flow direction. We compare the directional demand to the available capacity in that direction to determine if demand exceeds capacity in that particular direction. Further, we describe how to use directional capacity analysis to derive *capacity reduction maps* that can be applied to Flow Constrained Areas (FCA) determination and other ATM applications.

5.1 Introduction

Estimating the capacity of an airspace is fundamental to ATM. If the demand for an airspace exceeds the capacity, then a TFM strategy must be implemented. Demand for an airspace is determined by the number and type of aircraft that desire to fly through the airspace during a given period of time. However, the demand may arrive at and pass through an airspace in a variety of directions; thus, we quantify in this work the dominant direction of travel for the demand — essentially, the direction for which the majority of aircraft in a flow are traveling. The capacity of an airspace is loosely defined as the maximum number of aircraft per unit time that can be safely accommodated by the airspace given controller and pilot workload constraints as well as airspace constraints. The primary concern of this chapter is the effect of convective weather constraints on directional airspace capacity, with capacity determined by the geometric constraints induced by hazardous weather (rather than by controller workload constraints).

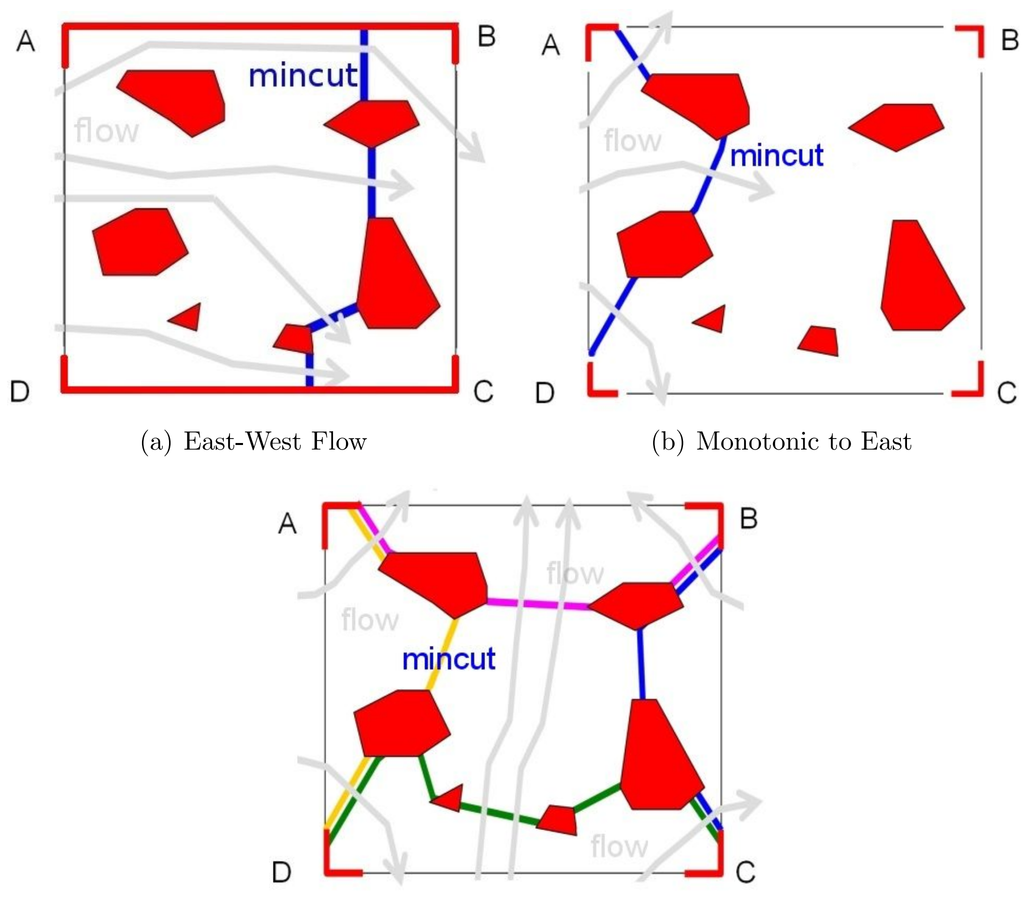
The weather has a huge influence on the performance of the NAS. The number of weather-related delays in the NAS is (approximately) twice the number of non-weather delays [26, 25]. The Aviation Capacity Enhancement Plan [16] lists weather as the leading cause (65% to 75%) of delays greater than 15 minutes, with terminal volume as the second leading cause (12% to 22% of delays). Thunderstorms account for approximately 50% of the weather-related delays, low visibility 35%, and heavy fog the remainder [15]. Because weather is such a major factor limiting capacity, we focus on the relationship between capacity, direction of demand, and weather severity.

The operational conditions considered in this work are motivated by the NextGen [21]. We consider the estimation of capacity in a future ATM system where aircraft are not required to follow jet routes. Rather, we consider operational rules for en route airspace where traffic finds routes through the gaps in weather constraints as we have previously described in the literature [28]. As described in chapter 1, the capacity of an airspace is defined as the maximum number of non-interfering air lanes that can be safely packed amidst constraints.

5.2 Motivation

From previous research [28], we have determined that airspace capacity is dependent on the ATM flow structure, which may specify that flow is in a particular direction, e.g., East to West, or that it is in all directions, e.g., in Free Flight [42]. Directional capacity is derived using a geometric and continuous version analogous to the standard max-flow/min-cut theory from network analysis [1]. As described in chapter 2, the (discrete) continuous max-flow/min-cut theorem states that the maximum number of pairwise-disjoint thick paths is equal to the shortest $T-B$ path length in the *thresholded critical graph* of the domain, and we refer to this $T-B$ shortest path as the *mincut* of the domain.

The mincut can be applied to different ATM flow structures [28] as shown in Figure 13. If the flow is unidirectional, for instance, from West to East, then the mincut is determined by allowing the source to be line segment AD and the sink to be line segment BC. If the flow is monotonic to the East, then the mincut is applied from point A to point D. If the mincut is applied to a Free Flight traffic flow, in which aircraft may fly in any direction, then the maximum flow computation involves computing mincuts for various combinations of sources and sinks defined by pairs of corners A, B, C, and D [28]. Each one of these mincut computations returns the theoretical bottleneck to the flow (the mincut) under the different ATM flow structures (uniform flow, monotonic flow, or Free Flight) [28].



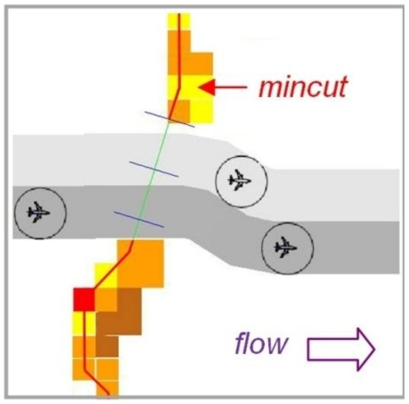
(a) East-West Flow

(b) Monotonic to East

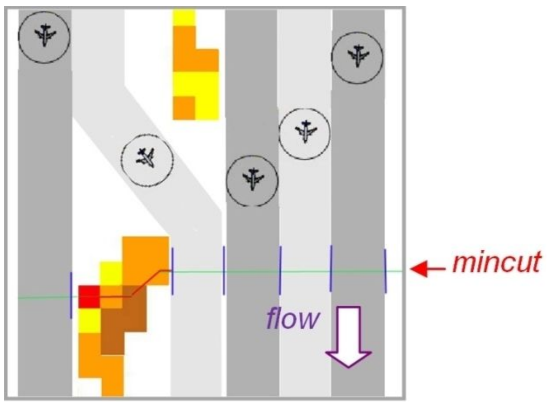
(c) Free Flight (All Directions)

Figure 13: The mincut algorithm determines the maximum permeability of an airspace under three different ATM flow structures. The boundary segments and inner polygons colored in red are obstacles.

Capacity is a function of the direction of travel of the flow; this is easy to see based on analyzing a squall line convective weather system, as illustrated in Figure 14. In this example, the mincut for a flow from West to East is compared to a mincut from North-South, each case with the same required air lane width. The mincut reveals that two lanes of traffic are the maximum that can pass from West to East, but five lanes of traffic is the maximum that can pass from North to South. Thus, the directional capacity (sometimes referred to as the directional permeability) of the airspace is maximized if the flow is allowed to pass from North to South, or from South to North (or some combination of both). From this we see that it is important to study the directional permeability, and to compare it to the direction of the dominant flow as determined by the demand profile for a given period of time that corresponds to the forecast time. If the dominant flow in this example were from East to West, then even though greater capacity may be achieve by a North to South flow, this may not service the demand.



(a) West-East Flow (Maximum of 2 air lanes)



(b) North-South Flow (Maximum of 5 air lanes)

Figure 14: The permeability of a squall line with respect to two directions of flow.

5.3 Approach

There are two aspects to this problem that we pursue:

- Direction of Dominant Demand
- Directional Permeability

Next, we describe our approach to each of these.

5.3.1 Direction of Dominant Demand

We consider two types of demand: filed flight plans for todays NAS (e.g., based on Enhanced Traffic Management System (ETMS) data), and 4D Trajectories (4DTs) for the NextGen. In either case, the dominant demand transforms the trajectory data into a set of waypoints, with time stamps at each waypoint, and projects those data points onto the boundary of the directional capacity kernel (square and circular kernels, as described in the next section). The intent of this analysis is to characterize the magnitude and direction of demand. We are particularly interested in understanding how aircraft approach a capacity-constrained airspace, since, as we presented above, the capacity of an airspace depends on the dominant approach direction. We refer to this as *demand-driven capacity.*

The reference direction $\boldsymbol{d}$ (a unit vector) is established over a look-ahead time period consistent with the look-ahead weather forecast time period used in the directional permeability analysis (described later). We look at a look-ahead time T_{forecast} to $T_{\text{forecast}} + \Delta T$. For now, we simply say that $\boldsymbol{d}$ will be defined by one of 8 standard directions, North (N), East (E), South (S), West (W), NE, NW, SE, and SW. Examples are given below for several cases of the dominant direction of flow. Later, these flow directions are compared to the directional permeability. When comparing the directional demand to the directional capacity; we quantify the total directional demand in a given direction from the equation:

$$D = \frac{1}{N} \sum_{i=0}^{N} \max(0, d_i \cdot \boldsymbol{d})$$

for N trajectories passing through the airspace, with d_i representing the unit tangent direction of travel for the i^{th} trajectory, and $\boldsymbol{d}$ representing the unit vector airspace reference direction. In this definition, the max( ) function ensures that only positive components are incorporated. The negative components from each flight translate into positive demand contributions in some other direction, so a full accounting is made without the implied redundancy. The metric applies to traffic crossing an arbitrary polygon boundary defined at a flight level, as illustrated in Figure 15.

We quantify the directional demand primarily on square and circular kernels (described later); however, the metric applies to any polygon boundary (e.g., sectors). The directional demand will be compared to the directional capacity, so both are evaluated for the same time period of interest from T_{forecast} to $T_{\text{forecast}} + \Delta T$.

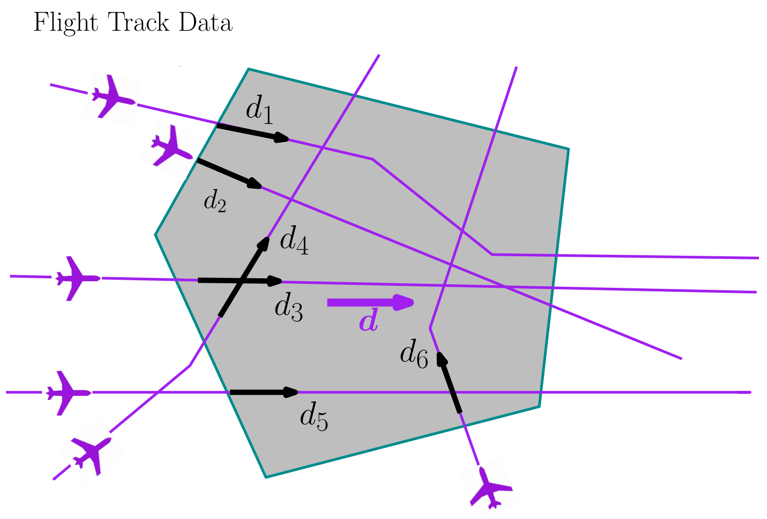


Figure 15: A dominant flow direction defined by filed flight plans and a polygon boundary.

A variant of the above dataset can be generated for the NAS using current time data, and making near-time projections based upon the position and velocity of each flight. Hence, if flight plans change because of weather conditions, tracking continues with updated data. In our effort, we do not build a trajectory prediction algorithm to estimate the aircraft that are expected to fly into a given square kernel, circular kernel, sector, or center boundary; rather, we assume some other algorithm or system will provide us these data.

Several examples (Figure 16 through Figure 18) of the dominant demand function demonstrate how in NextGen, traffic flows may result in dominant flow directions. If there is *no* dominant flow, for instance in the case of Free Flight from all directions of travel, then the dominant demand is low, as illustrated in Figure 17. If weather forces traffic to flow in a particular direction, as illustrated in Figure 18, then the dominant demand will be maximized at that particular direction.

Note that we also quantify the *net* demand, which is simply the dominant direction parameter multiplied by the total number of flights N. In ATM applications, strategies will be developed for distributing this demand into other directions when certain other directions have already reached capacity.

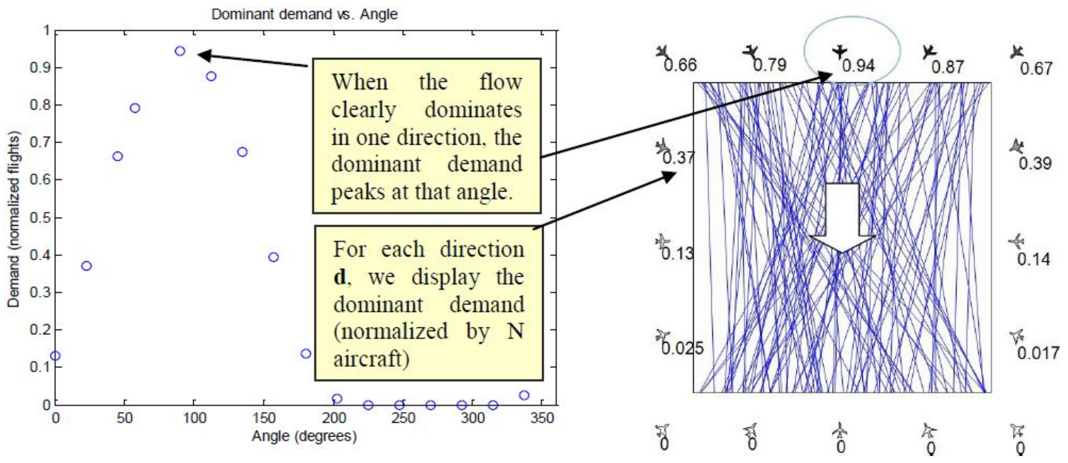


Figure 16: A dominant southern flow direction defined by filed flight plans and a square boundary.

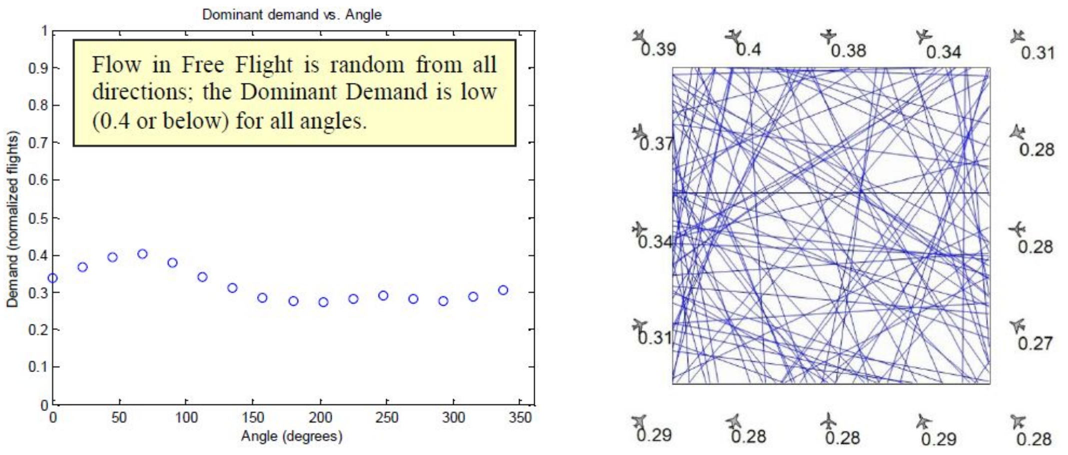


Figure 17: A weak uniform dominant demand defined by filed flight plans and a square boundary.

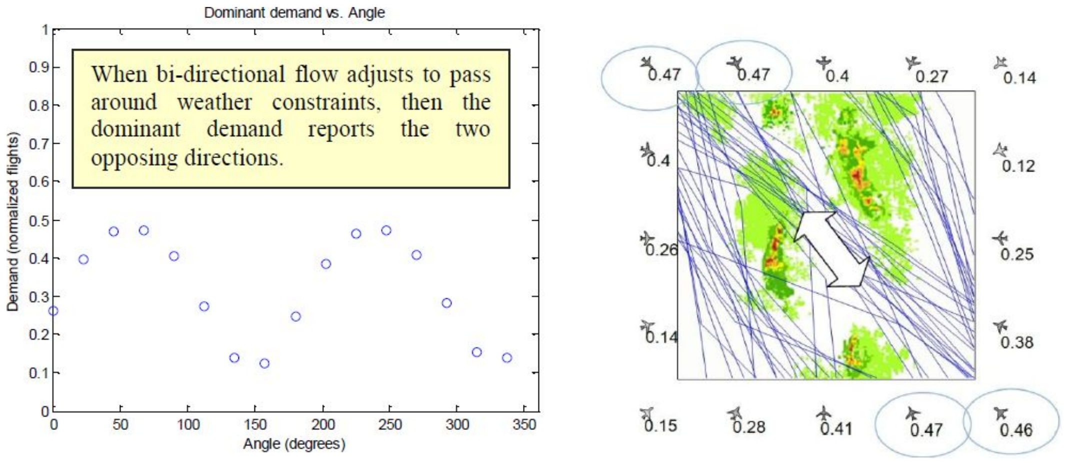


Figure 18: Two dominant flow directions defined by filed flight plans and a square boundary.

5.3.2 Two Methods for Directional Permeability

There are two types of kernels that we investigated:

- Square Kernels
- Circular Kernels

First, we discuss square kernels, then circular kernels. In general, further application of the mincut applied to other boundaries (e.g., sector boundaries) is found in the literature [28, 37, 45, 46, 47]. An ATM application also may be applied to ensemble forecasts in the form of the square kernel [48].

The square kernels are as follows. These mincut filters are either bounded by points (red circles) or impenetrable walls (red line segments) which artificially constrain the flow (arrows) across the grid as shown in Figure 19. The directions of the filters are along eight fundamental directions of travel: N, S, E, W, NW, NE, SW, SE, and all directions (we call this Free Flight) [28].

The computation of the mincut and mincut properties for assessing throughput within the kernel are as follows. In Figure 20, the mincut is from a Northern bounding surface to a Southern bounding surface as shown. This figure shows some of the segments (dashed blue) used in constructing that mincut; these segments are a part of the critical graph, and the mincut is a particular part of the critical graph that connects the boundary conditions. Note that all lines connecting either the upper or lower surface are perpendicular to that

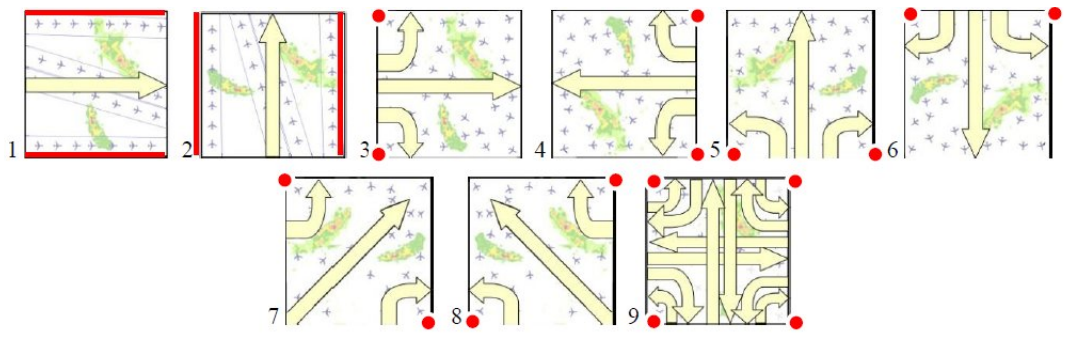
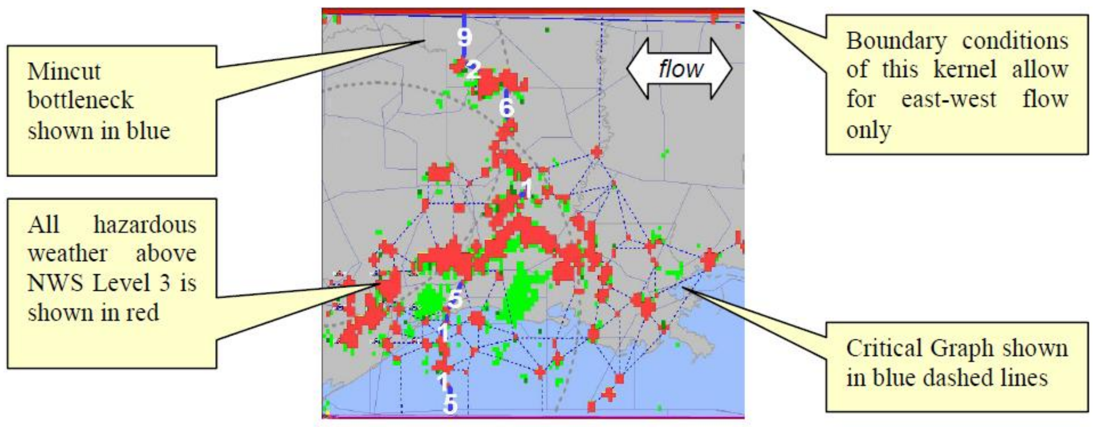


Figure 19: A set of 9 grid-based mincut filters.

boundary condition (the shortest distance between the boundary condition and the weather hazard). Also note how only the red polygons are relevant as they represent weather that exceeds some threshold value (which the green polygons do not). Numbers labeled on the mincut (colored in blue) indicate the total number of lanes of traffic that can pass through the mincut, assuming that each lane of traffic has a RNP requirement that is specified as an input to the algorithm.


Mincut bottleneck shown in blue
All hazardous weather above NWS Level 3 is shown in red
flow
Boundary conditions of this kernel allow for east-west flow only
Critical Graph shown in blue dashed lines
Figure 20: The mincut identifies the bottleneck to the flow, and is constructed by searching for the critical graph (dashed blue lines) and identifying the minimum distance path from one boundary condition to the other boundary condition.

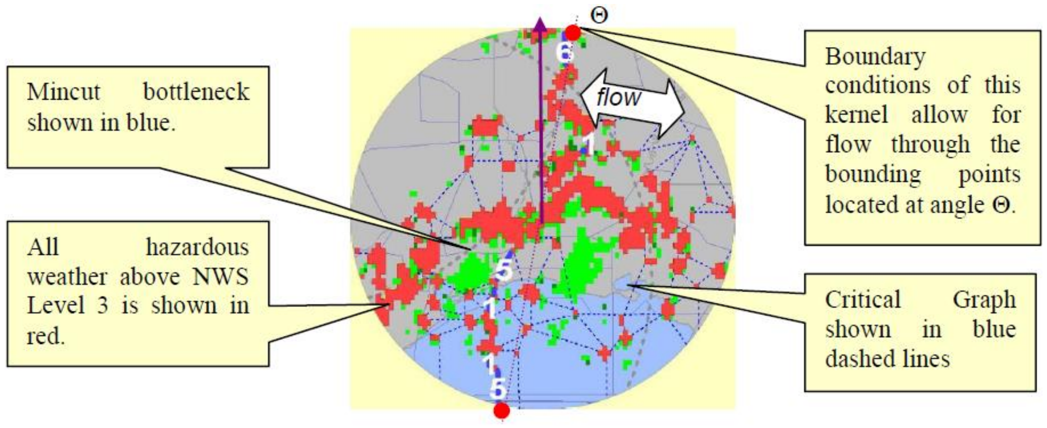

Θ
Mincut bottleneck shown in blue.
All hazardous weather above NWS Level 3 is shown in red.
flow
Boundary conditions of this kernel allow for flow through the bounding points located at angle Θ.
Critical Graph shown in blue dashed lines
6
1
5
1
1
5

Figure 21: The mincut identifies the bottleneck to the flow using a circular boundary condition for the kernel.

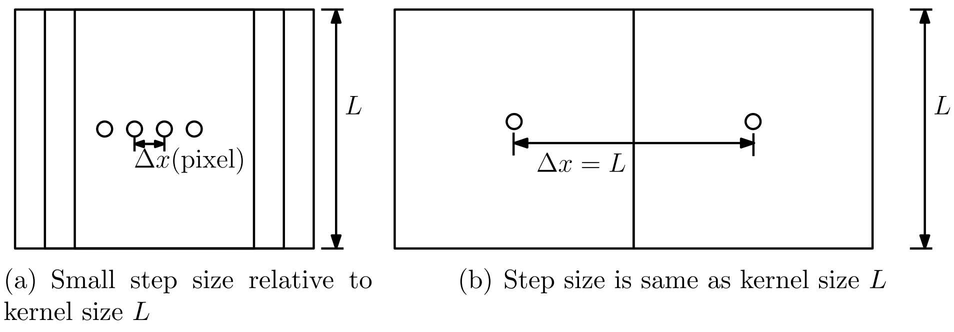
A second mathematical model that we are investigating is based on a circular kernel. In this model, the key parameters are the radius R of the circular filter and the orientation θ. The mincut properties for assessing throughput are as follows. In Figure 21, the mincut (blue) is from a top bounding point (red circle) to a bottom bounding point (red circle). The line from the top point to bottom point is at any arbitrary angle θ. For a point p, the directional capacity in the flow direction Θ (flow direction Θ is orthogonal to the filter direction of θ) is given by computing the discrete mincut (number of air lanes) between the goal posts positioned at distance R from p with orientation θ. The dashed blue segments used in constructing that mincut are the critical graph, and the mincut is a particular part of the critical graph that connects the boundary conditions. Numbers labeled on the mincut indicate the total number of lanes of traffic that can pass through the mincut, assuming that each lane of traffic has a RNP requirement. The circular kernel is applied at each of several discrete choices of direction θ, for any specified radius R. This yields a directional capacity, capacity(p, R, θ), which can be computed for each point p of the domain of interest.

Finally, whether a square kernel is used or a circular kernel, there is the issue of the size of the kernel (length of a square kernel side L or radius R of the circular kernel) and the step size of how you analyze weather constraints in the latitude and longitude (or any other) direction. We studied two extremes:

- Pixel-by-Pixel step sizes here the center of the kernel (square or circular) is applied pixel-by-pixel; that is, the Δx and Δy step size for locating where the kernel center is applied is the same size as a pixel of weather data independent of the kernel size (radius R or square side length L); or

- Kernel-by-Kernel step sizes here the center of the kernel (square or circular) is applied such that each kernel is adjacent to the previous kernel; that is, the Δx and Δy step size for locating where the kernel center is applied is the radius R or square side length L of the kernel.

The modeling of small Pixel-by-Pixel step sizes vs large Kernel-by-Kernel step sizes is depicted in Figure 22.



(a) Small step size relative to kernel size L

(b) Step size is same as kernel size L

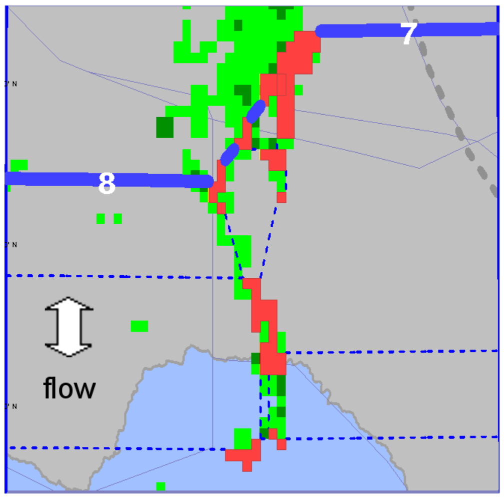
Figure 22: Small Pixel-by-Pixel vs large Kernel-by-Kernel step size increments (square side length L).

5.4 Examples

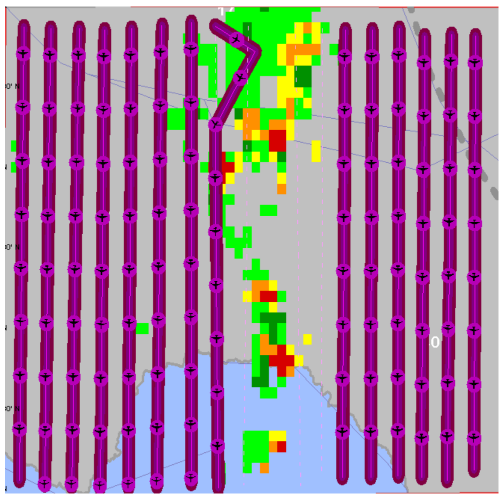
In this section, we present a series of examples to demonstrate the concepts of directional capacity.

5.4.1 Mincut Capacity vs Dominant Demand Direction

In Figure 23 and Figure 24, a slow-moving squall line is considered. North-South flow is virtually undeflected because the weather is so slow relative to the flow velocity. The mincut for the hazardous weather is shown on the left. The flow from an algorithmic solution to the route planning problem is shown on the right. In Figure 23, one flight takes advantage of the slight weather movement with a zigzag designed to consume time while an additional lane of traffic opens. This mincut applies for a single instance in time, so the number of flight paths passing through the weather may not equal the number shown in these figures.



(a) A mincut of 15, shown in solid blue



(b) Weather-avoidance routes

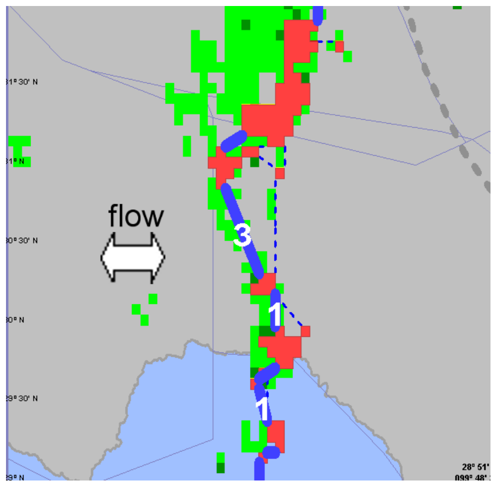
Figure 23: The mincut throughput for North-South flow and an algorithmic approximation to a set of routes that approach the mincut capacity.

In Figure 24, the same slow-moving squall line is analyzed with a kernel that measures the capacity for East-West flow. East-West flow allows just three paths to pass through the weather. Again, this is a dynamic situation, and even though the instantaneous mincut on the left indicates two breaks in the squall line at the south, these gaps close up and do not allow continuous flow to pass through. Analyzing the mincut over a given period of time is needed to establish the capacity, not just the mincut at a given time, since the weather hazard is continuously changing is shape and size.

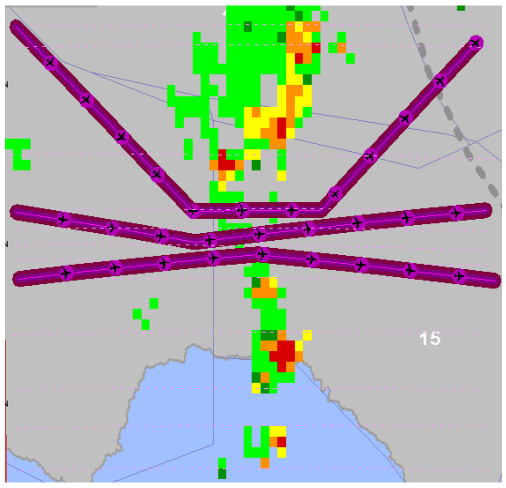
The point of this simple squall line example is to demonstrate the fact that a squall line can have a significant difference in throughput in the two fundamental directions: in line with the squall line vs orthogonal to the squall line. This is not surprising, and is clearly what is indicated by the mincut result and the flow planning algorithm.

5.4.2 Square Kernel Examples

In Figure 25 we process convective weather data using several different directional permeability kernels for different directions and for different kernel



(a) A mincut of 5, shown in solid blue



(b) Weather-avoidance routes

Figure 24: The mincut throughput for East-West flow and an algorithmic approximation to a set of routes that approach the mincut capacity.

size, using the same weather data for all filters. Figure 26 through Figure 27 demonstrate the results.

In Figure 26 we note that the directional permeability results are quite different in comparing the mincut throughput for each direction. However, we need to note that in order to compare these results to one another, the results for East-West and North-South flow are comparable, and the results for diagonal flows are comparable, but we must divide the results for diagonal flows by $\sqrt{2}$ in order to compare diagonal flows to East-West and North-South flows. It is thus useful instead to represent results in terms of reduced capacity relative to the clear weather capacity in the particular direction of interest.

5.4.3 Circular Kernel Example

Our results demonstrate how the mincut throughput for a specific center point p is a function of the angle of the circular kernel flow direction. In contrast to the grid-based kernel, where there were eight fundamental directions for the kernel, in the circular kernel, we can run the kernel for any angle Θ of flow.

In this example based on real weather data, we demonstrate the results for a constant RNP-2 and kernel radius 150 nmi, with the direction of flow varying in 1° increments from $\Theta = 0°$ to $\Theta = 180°$ (angles are defined according to the Polar coordinate system, where 0° direction is to the East, 90° is to the

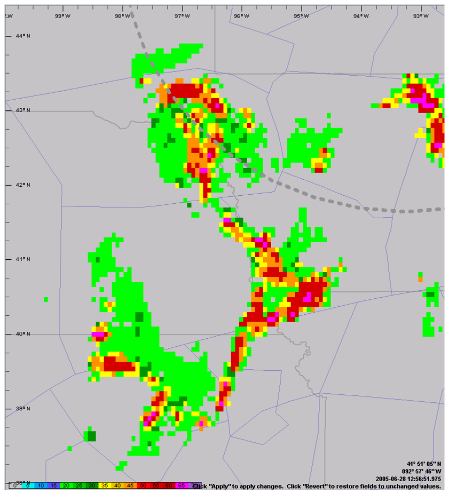
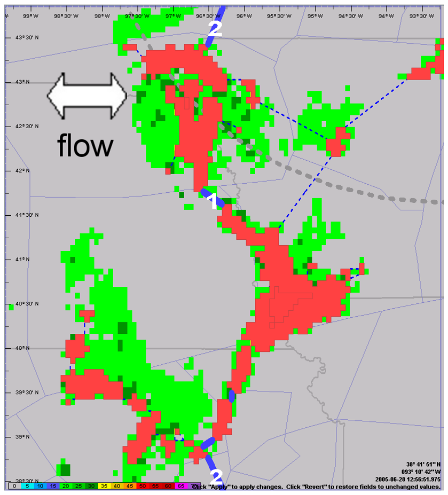


Figure 25: Convective weather data used for analysis (6/28/05, 13:00Zulu). Convective Weather is thresholded at NWS Level 3 or above to identify weather hazards.

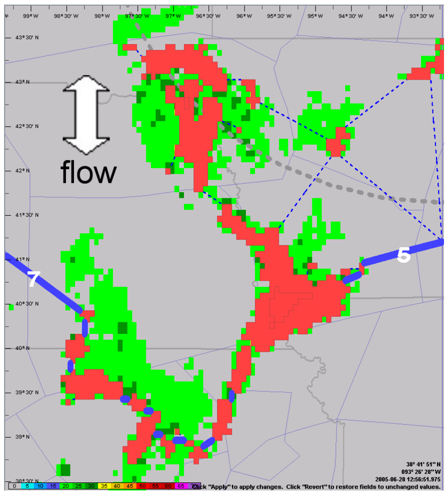
North, 180° is to the West, etc). For any angle of flow direction Θ, we place two boundary conditions (the red disks in Figure 28) as obstacles at two ends of the $\Theta + 90°$ diameter of the circular kernel. We demonstrate in Figure 28 that if the direction of flow of traffic is to pass across these weather hazards, how the mincut and the capacity (expressed in terms of the total number of air lanes that can cross between the boundary conditions) change as we vary the direction of flow Θ; we only show angles at which there is major "structural change" (defined as change in the topology of the mincut in the critical graph except for the two edges adjacent to the two boundary conditions) in the mincut and refer to those angles as the *critical angles*.

We plot our directional capacity data as a function of the angle of flow direction in 1° increments, as illustrated in Figure 29. The directional capacity plot exhibits a bimodal pattern, suggesting that the flow direction is favored at $\Theta = 162°$ and $\Theta = 15°$, both approximately aligned with West-East direction; the directional capacity is minimized at $\Theta = 107°$, suggesting that North-South flow direction has substantial capacity reduction and should be avoided if possible. This set of scenarios and the resulting directional capacity as a function of dominant flow demand indicates a need for a capacity analysis capability for NextGen that considers directional capacity.

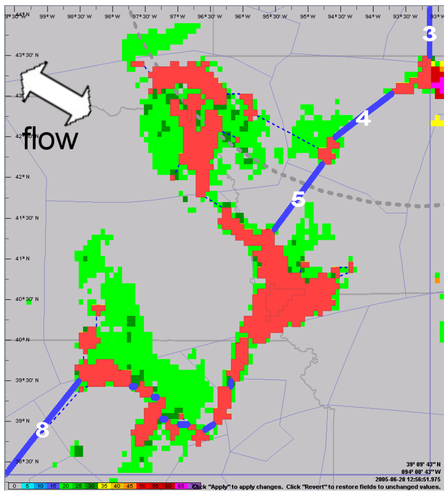
These results indicate that there may be an application for the circular kernel in determining a gate for traffic passing through a weather constrained



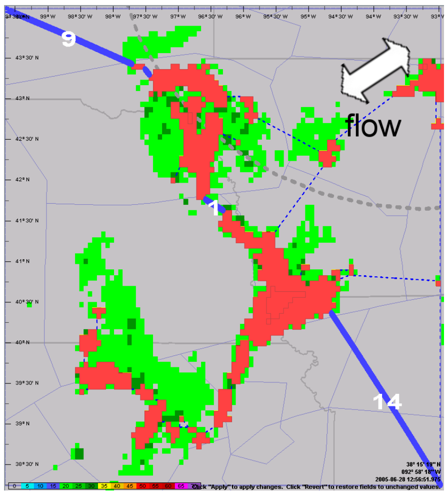
(a) 5 Air Lanes Throughput



(b) 12 Air Lanes Throughput

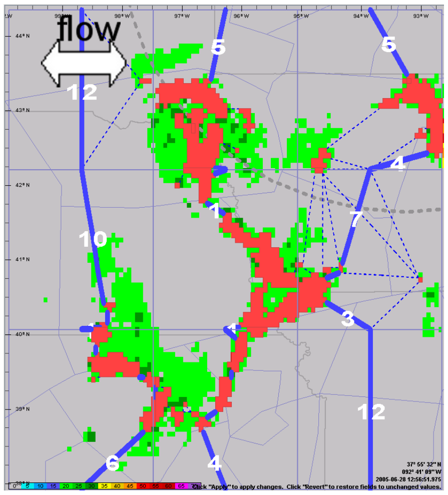


(c) 20 Air Lanes Throughput

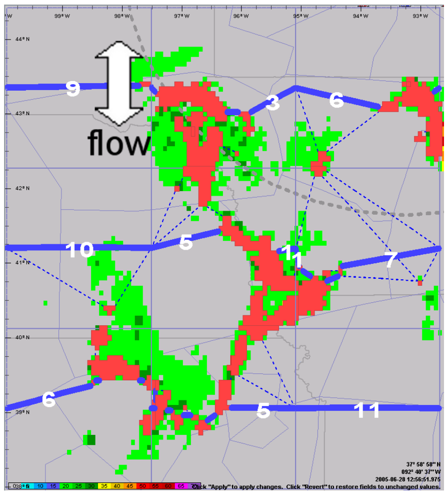


(d) 24 Air Lanes Throughput

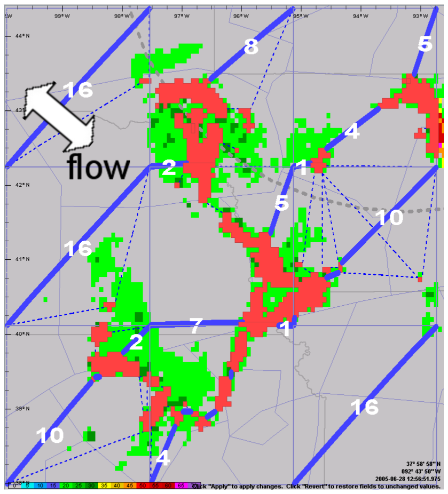
Figure 26: Mincut results as a function of flow direction (6/28/05). 240 nmi Grid; Minimum Gap Size of 10 nmi Required.



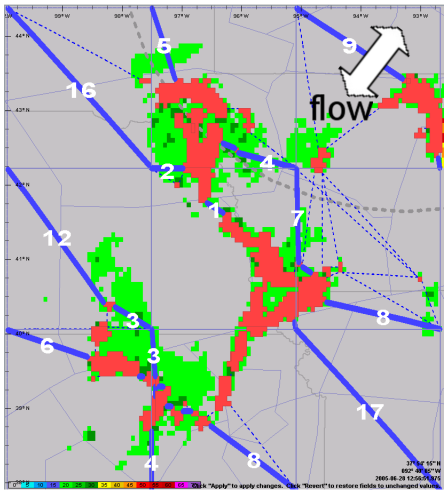
(a)



(b)

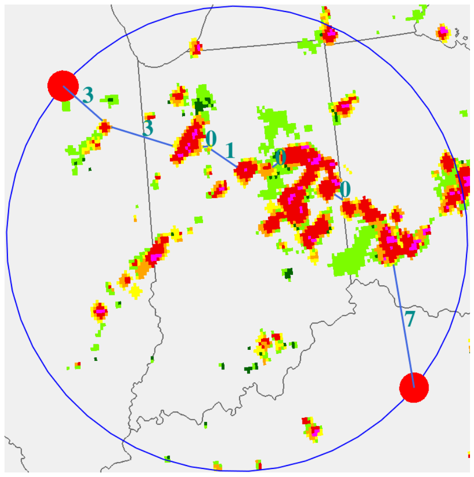


(c)

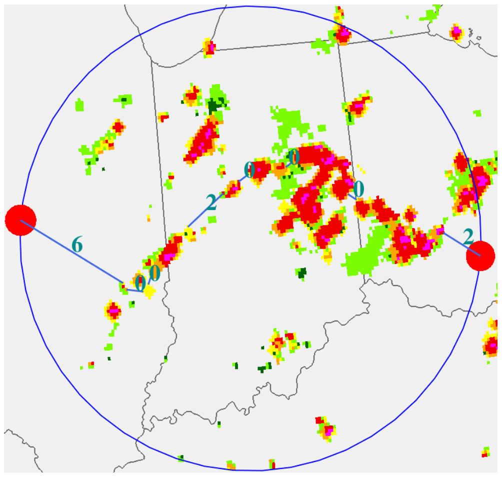


(d)

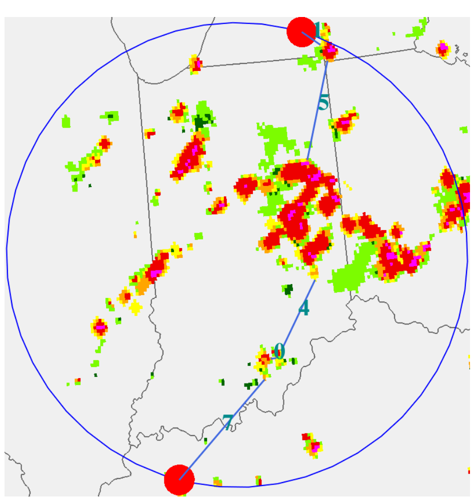
Figure 27: Mincut results as a function of flow direction with large kernel size (6/28/05). 120 nmi Grid; Minimum Gap Size of 10 nmi Required.



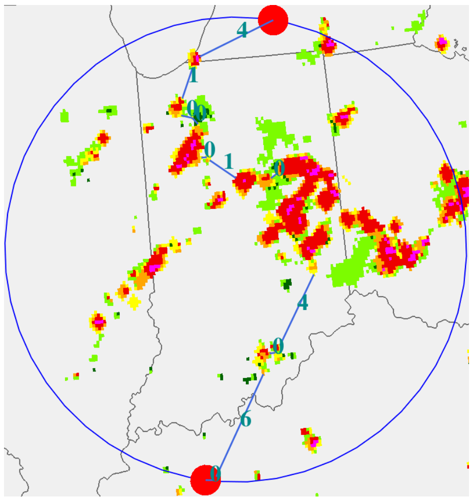
(a) $\Theta = 44°$, Capacity 14 air lanes



(b) $\Theta = 80°$, Capacity 10 air lanes



(c) $\Theta = 159°$, Capacity 17 air lanes



(d) $\Theta = 166°$, Capacity 16 air lanes

Figure 28: The mincut and the capacity are plotted at critical angles. Flow directions are perpendicular to the diameter connecting the two goal posts. Convective weather is thresholded at NWS level 3 or above (colored as yellow or more severe) to identify weather hazards.

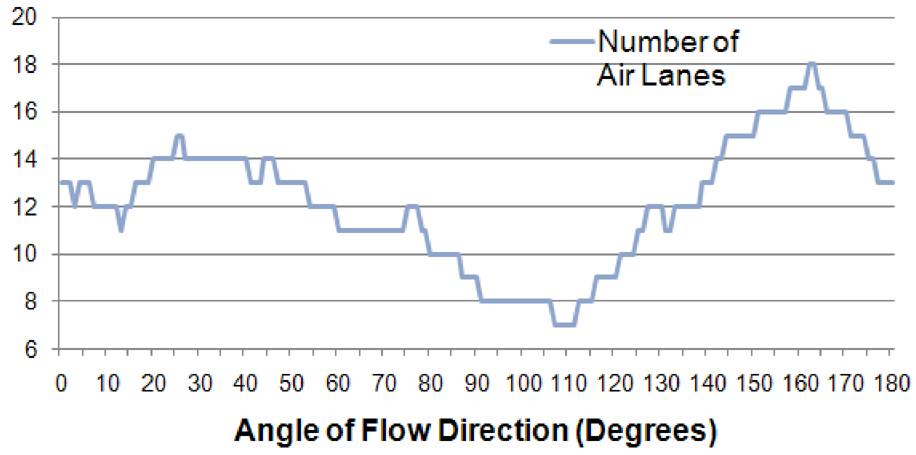


Figure 29: The directional permeability is a function of the flow direction.

region. In many cases today, a Flow Evaluation Area (FEA) is placed along center or sector borders, and it is possible to place the two boundary conditions of the circular filter (the red points) at points at the two extremes of an FEA. The mincut then estimates the capacity for flow crossing between the two points, allowing controllers to use this technique as a quick measure of capacity reduction. If the capacity reduction is substantial, this is indicative of the need for a TFM strategy.

5.5 ATM Applications

Capacity Reduction Maps The results from the eight fundamental directions of flow from the grid-based kernel or from the angular flow directions of the circular kernel can be used to define a capacity reduction map. Capacity reduction maps are relevant with respect to all forms of weather hazards, including convection, turbulence, and icing. Capacity reduction maps may be used to inform Air Traffic Service Provider (ATSP) or airline users of potential areas to be avoided in strategic planning of flights (used for informational purposes), or they may be used to set up the analytic determination of TFM plans, for instance, where to locate Flow Constrained Areas (FCA).

A capacity reduction map, for instance as shown in Figure 30, gives for each point p of the map the percent reduction (compared to clear weather) in capacity (in terms of number of air lanes), in a given direction Θ and for a given kernel (and kernel radius R) centered on the point p. A black pixel indicates 100% capacity reduction; levels of gray indicate percentage reduction

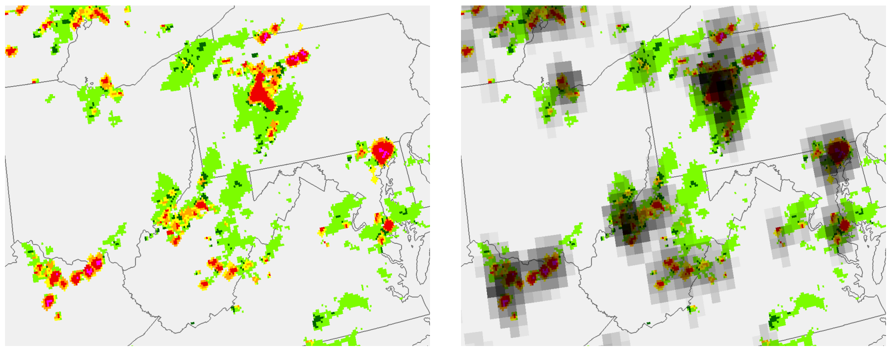

(a) NWS Reflectivity Weather (b) Capacity Reduction Map

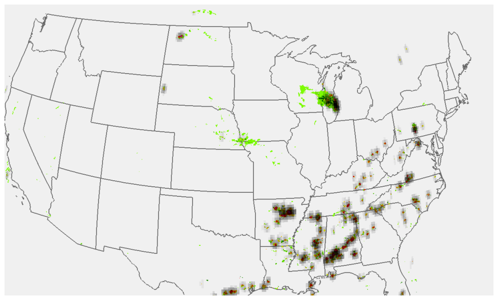
Figure 30: Circular kernel results for capacity reduction (North-South Flow, RNP 1, kernel radius R= 20 nmi, and grid spacing of 10.77 nmi).

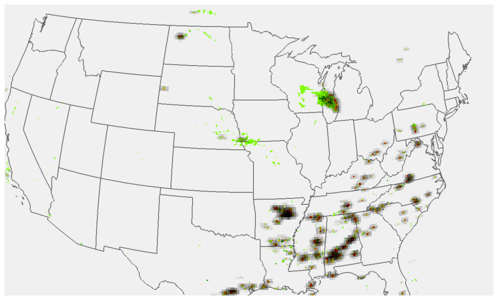
of capacity.

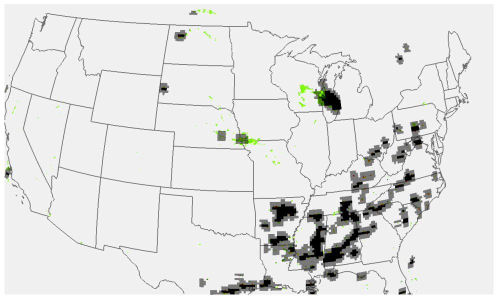
In Figure 31, the capacity reduction map is shown for two flow directions, North-South and East-West, under two different RNP values (RNP-1 and RNP-5). In general, as the RNP value goes up and one requires that the gaps between the weather constraints is larger, there is a more pronounced effect in the capacity reduction map. This occurs in both directions studied.

TFM Optimization Another ATM application of directional capacity is for NextGen TFM optimization. Researchers studying how to optimize TFM for NextGen need to experiment with techniques for TFM that optimize performance of the NAS. An example is where optimal TFM strategies [4] are planned on a grid lattice spanning the CONUS.

In such a formulation of TFM optimization, the local directional permeability must be established for the capacity of one grid cell with respect to traffic arriving into it from adjacent grid cells (Figure 32). The capacity is a function of the direction of the grid cell, in the eight fundamental directions: N, S, E, W, NW, NE, SW, and SE. Furthermore, if an alternating altitude rule is used, then the acceptable direction of flow is also a function of altitude. The grid-based capacity reduction described by the mincut for the applicable optimization grid size (e.g., 30, 40, or 50 nmi grid sizes) and in the direction of flow that links grid cells together would be used in the grid-based solution.


(a) North-South (RNP-1)


(b) West-East (RNP-1)


(c) North-South (RNP-5)

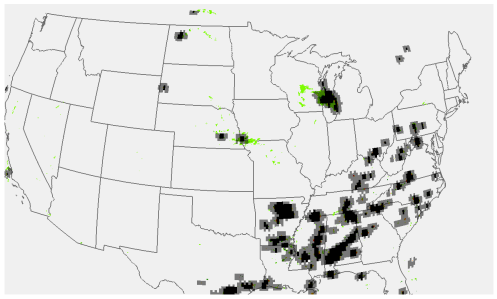

(d) North-South (RNP-5)

Figure 31: Circular kernel results for capacity reduction (East-West Flow vs. North-South flow, RNP 1 and 5, kernel radius R=20 nmi, and grid spacing 10.77 nmi).

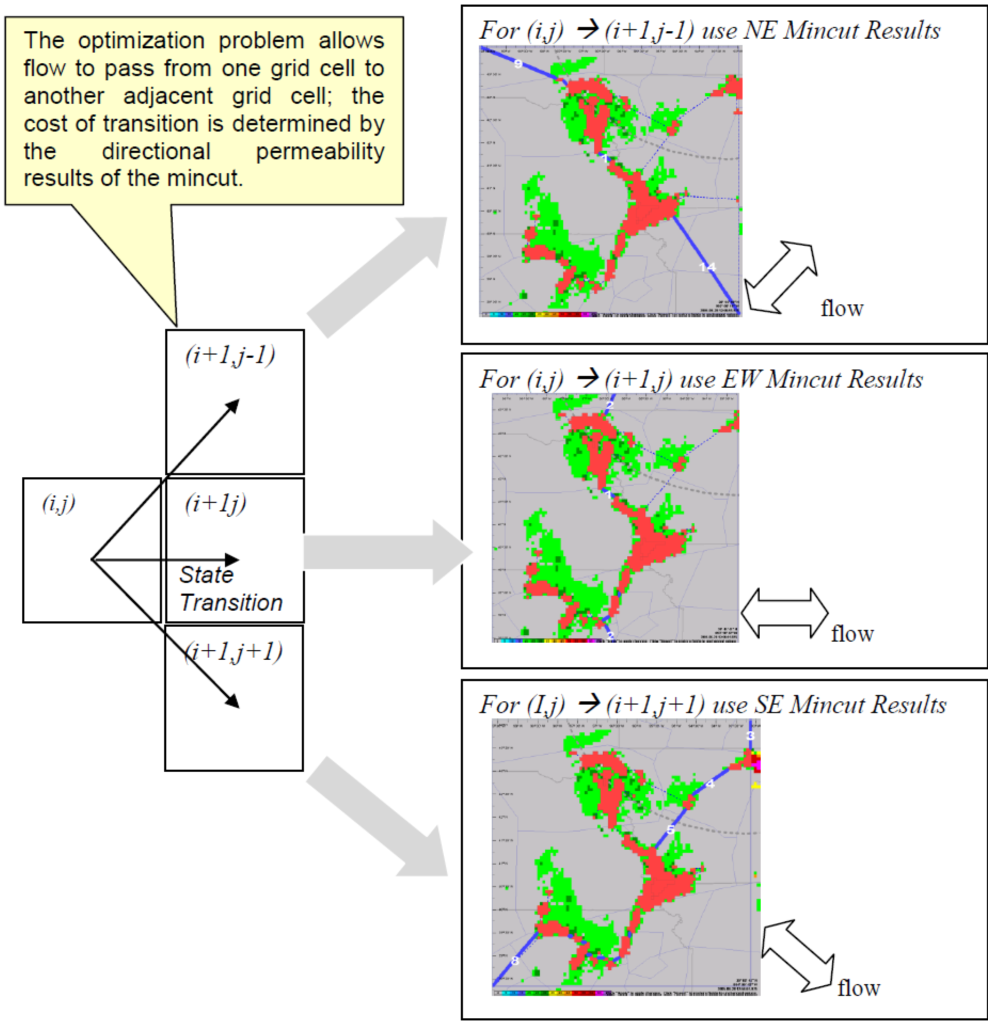

The optimization problem allows flow to pass from one grid cell to another adjacent grid cell; the cost of transition is determined by the directional permeability results of the mincut.
(i,j)
(i+1,j-1)
(i+1j)
(i+1,j+1)
State Transition
For (i,j) → (i+1,j-1) use NE Mincut Results
For (i,j) → (i+1,j) use EW Mincut Results
For (I,j) → (i+1,j+1) use SE Mincut Results
flow
flow
flow

Figure 32: Concept of directional capacity reductions used in TFM optimization.

5.6 Conclusion

This study clearly demonstrates that demand may be characterized as a directional demand, and capacity may be characterized as a directional capacity. Weather hazard maps may be transformed into capacity reduction maps, based on a formal mathematical transformation that is referred to as the Mincut, a concept from computational geometry with associated parameters (RNP size, grid or circular kernel size, and flow direction). The mincut can be applied to a square kernel or a circular kernel in order to identify the capacity reduction in the vicinity of a point (in the center of the square or circular kernel) for a given direction of flow. Capacity reduction kernels may be designed for fundamental directions (N, S, E, W, NW, NE, SW, SE) for square kernels, or for an arbitrary angle (Θ) for circular kernels. Capacity reduction kernels may be convolved with weather hazard maps in order to arrive at the total capacity reduction map for the NAS. TFM solutions must consider how to direct the traffic flow to conform with the limits of the capacity in the direction of the flow the directional demand is limited by the directional capacity, and if the demand exceeds the capacity in a given direction, the flow upstream must be diverted to some other location where there is excess capacity. Future NextGen TFM optimization algorithms may thus automate this process when formulating "outer loop" TFM optimized solutions that can reason about the amount of flow to pass in the particular directions that maximize capacity for the NAS as a system.

In the future, we plan to investigate, algorithmically, how to compute all the critical angles in a circular kernel without discretizing the 360° angle. For any goal post locations, p_i, p_j, if they are directly connected to the mincut via obstacles H_i and H_j, respectively, the mincut length is $d(p_i, H_i) + d(p_j, H_j) +$ Constant where the constant is the shortest path distance between H_i and H_j in the critical graph and can be pre-computed. A distance function of this form might suggest the use of some kind of second order additively weighted Voronoi diagram for the obstacles $\mathcal{H}$ to compute the critical angles.

6 Flexible Tree Routing

This chapter represents joint work with Joondong Kim, Joseph S. B. Mitchell, and Jimmy Krozel, used with their permissions. In this chapter we consider the problem of routing aircraft in the transition airspace. Within the transition airspace, today's traffic obeying Instrument Flight Rules (IFR) follow tree-like structures used for Standard Instrument Departures (SIDs) as well as Standard Terminal Arrival Routes (STARs). The SIDs and STARs are based on fixed airway structures, defined by fixed navaids and fixed routes connecting navaids, and this current system does not adapt well to changes in hazardous convective weather and other constraints. We consider an alternative approach to transition airspace routing, in which we have the freedom to choose where to put flexible merge points instead of fixed navaids, and to choose how to connect the merge points to an optimized tree structure for routing.

We have shown that the flexible tree routing problem is in general NP-hard. However, when the number of "layers" (defined later in this chapter) in the search graph is small and can be treated as a constant, we have a polynomial time algorithm to solve the problem exactly, based on dynamic programming. We have implemented the algorithm in a system which we call *Tree-based Route Planner* (TBRP), and we demonstrate results of experiments performed using TBRP in section 6.4.

6.1 Introduction

6.1.1 Modeling

In this section we describe our model of airspace geometry, aircraft navigation performance requirements, and the specifications to the routing problem we are solving.

Airspace Geometry and Geometric Constraints In our ATM applications, the transition airspace is modeled as an annulus usually centered at an airport; typical radii of the inner and outer range rings of the annulus are 40 nmi and 150 nmi, respectively. We consider the airspace to have hazardous weather constraints (severe convective weather, severe icing, and severe turbulence) and Special Use Airspace (SUA) constraints where aircraft are not permitted to fly. The annulus is subdivided, typically into four quadrants in four directions (Northwest, Northeast, Southwest, and Southeast), with each quadrant designated for either arrival or departure traffic. In contrast to todays system where there would be one metering fix per arrival quadrant, in NextGen, for each quadrant, there are up to three metering fixes on its inner range ring whose locations are either fixed or dynamically determined by our algorithms, and any arrival flight may be metered by any of the metering fixes available. Figure 33 gives an illustration of the airspace geometry. In general, our weather avoidance routing solutions are defined for aircraft traveling from the transition airspace outer range ring to the metering fixes, or from departure fixes (not shown) to the outer transition airspace boundary. In this study, we are mostly concerned with designing routing solutions for arrival flights, but it should be noted that this particular case is algorithmically equivalent to the problem of designing routing solutions for departing flights.

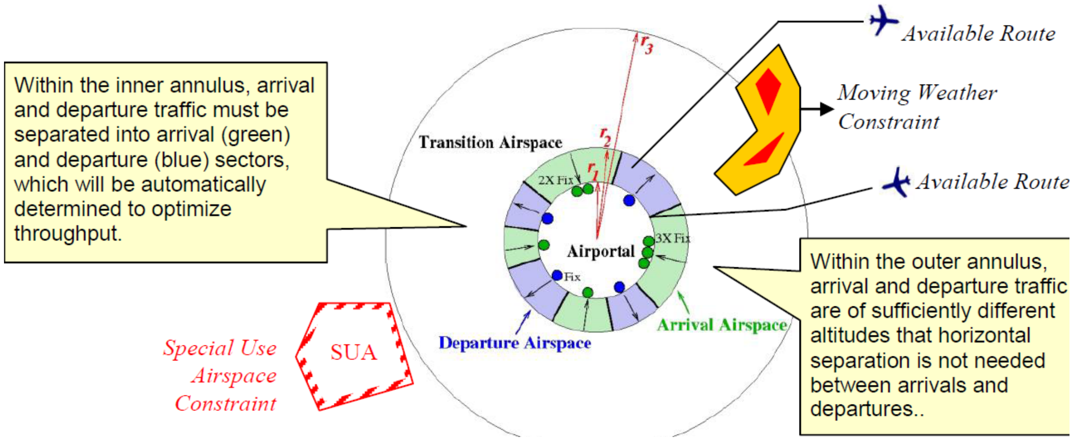

Within the inner annulus, arrival and departure traffic must be separated into arrival (green) and departure (blue) sectors, which will be automatically determined to optimize throughput.
Available Route
r_3
Moving Weather Constraint
Transition Airspace
r_2
r_1
2X Fix
Available Route
Airportal
3X Fix
Fix
Within the outer annulus, arrival and departure traffic are of sufficiently different altitudes that horizontal separation is not needed between arrivals and departures..
Special Use Airspace Constraint
SUA
Departure Airspace
Arrival Airspace

Figure 33: Transition airspace geometry with weather and SUA constraints.

RNP Requirements RNP-n specifies that the RNAV system is certified to stay within n nmi of the intended lateral (vertical) routing at least 95% of the time, including the time during turns. RNP requires that turns are executed at Trajectory Change Points (TCPs), and that the containment area is defined by a horizontal distance RNP_H (e.g., $\text{RNP}_\text{H} = 1$ nmi) from the centerline between TCPs (Figure 34). The vertical RNP_V requires that the aircraft is within a vertical distance (e.g., $\text{RNP}_\text{V} = 500$ ft) away from the flight leg defined by two TCPs 95% of the time (Figure 34).

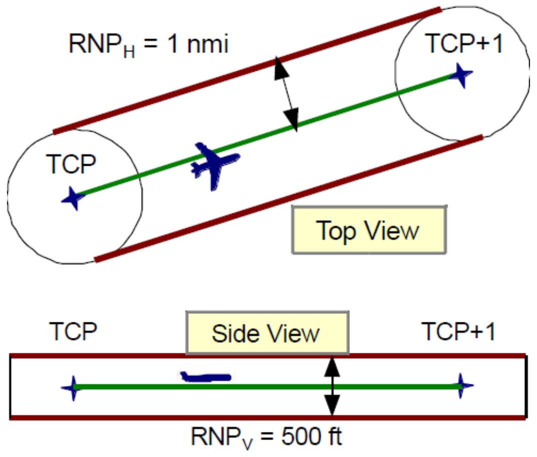


Figure 34: RNP requirements define a horizontal and vertical containment area.

For design purposes in the NextGen system, we double the RNP value to plan routes for which we expect the aircraft to be contained 99% of the time. Furthermore, we establish a required lateral separation distance, δ, (a safety margin, as illustrated in Figure 35) between any two non-intersecting RNAV routes, except at points where two route segments merge or branch, as in fan-in and fan-out trees. (Fan-in trees will be defined later in this chapter.) Thus, a RNP-n route ought to be considered as a weather avoidance route of width $4n$, with the additional constraint that any two such routes stay laterally separated by distance at least δ. However, for the simplicity of handling the additional routing requirement on lateral separation, we incorporate it into the requirement on route width, and we consider a RNP-n route as a weather avoidance route of total width $4n + \delta$ without any requirement on lateral separation.

Demand Profile In order to address the issue of having a continuum of possible entry point locations into the outer range ring, we subdivide the outer range ring of the quadrant into small arcs of equal length (usually around 10°) and treat each arc as a single entry point. In order to encode the demand profile, a tuple of all possible RNP values will be given (e.g., (RNP-1, RNP-2, RNP-3)). For each entry arc, there is a time series of tuples, with each tuple

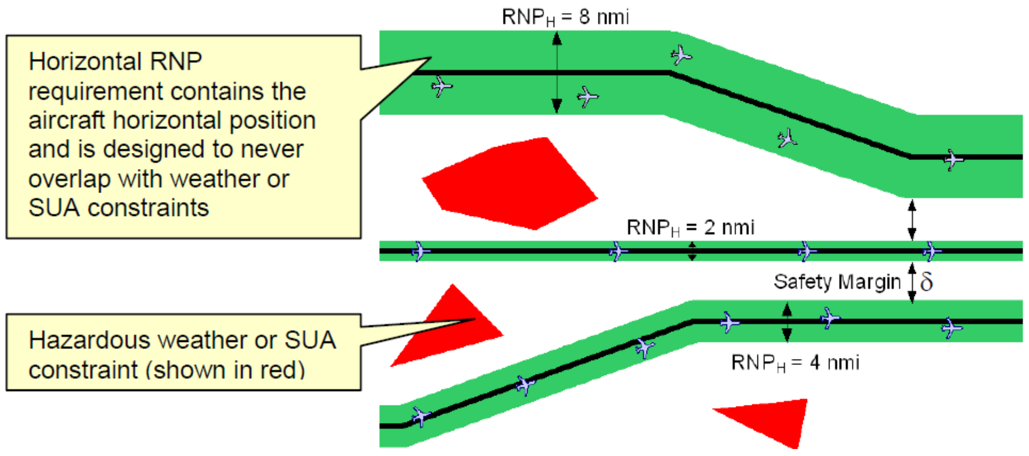

RNP$_H$ = 8 nmi
Horizontal RNP requirement contains the aircraft horizontal position and is designed to never overlap with weather or SUA constraints
RNP$_H$ = 2 nmi
Safety Margin δ
Hazardous weather or SUA constraint (shown in red)
RNP$_H$ = 4 nmi

Figure 35: Required minimum lateral separation between RNAV routes.

representing, within the corresponding time interval, the number of incoming flights for different RNP values that want to enter the quadrant using this arc (e.g., $(0, 5, 10)$ represents 0 flights with RNP-1, 5 flights with RNP-2, and 10 with RNP-3 wanting to use the arc); the time series data for all entry arcs constitute the demand profile.

Fan-in Tree We model the fan-in tree as a "fat graph embedding" of a tree (in the graph-theoretical sense), $T = (V, E)$, in the transition airspace, connecting entry arcs to the metering fixes. For each internal vertex of T, we require that its embedding stays at least distance r (e.g., $r = 5$ nmi) away from any hazardous weather or any other geometric constraints, and we call each such vertex as a *merge point*. For each edge e, its embedding is actually the centerline of a RNAV route and we require it to have certain weather clearance depending on the RNP value of the aircraft that fly along it; More precisely, for each edge e, we pre-compute its maximum clearance value, w_e, and allows a RNP-n aircraft to use it only if $w_e >= (4n + \delta)/2$. We also require that the edges dont intersect in the embedding, and that any path intersects only the embeddings of its endpoints.

We also impose operational constraints on the fan-in tree. There is a maximum in-degree bound, Δ, for the tree; we typically assume $\Delta = 2$ which means that at most two lanes of traffic can merge at any merge point, although our algorithms apply for any fixed value of Δ. We also impose a lower bound, L, on the length of any tree edge. The lower bound L models the minimum separation requirement between merge points so that traffic flows can be properly

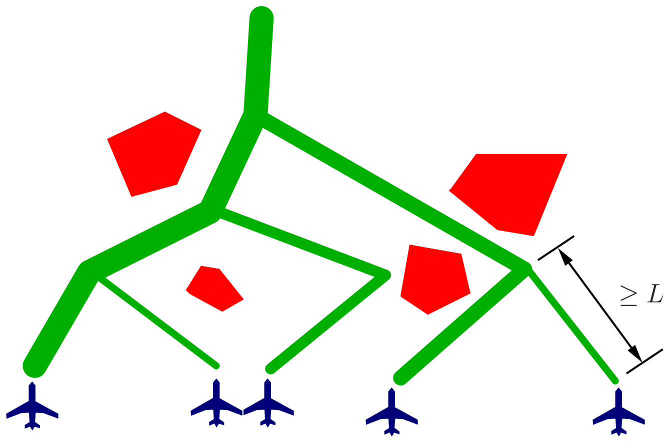


Figure 36: An example binary arrival tree with different thickness on different sub-paths.

organized; refer to Figure 36. Note that if there were not any lower bound on the length of edges, the maximum in-degree constraint would become irrelevant, since we can simulate a merge point of high degree by having many low-degree merge points arbitrarily close to each other.

Objective Function The objective function could be only depending on the number of flights accommodated at each entry arc, in which case we usually want to maximize the total number of arrival flights accommodated, or on both the number of flights and the total flight distance for the flights, in which case we want to first maximize the number of accommodated flights and then minimize the total flight distance. In fact, as long as the objective function can be computed recursively in our dynamic programming algorithms (shown later), the same algorithms could be applied to it.

6.1.2 Problem Formulation

Search Graph We only consider a quadrant of the annulus which models the transition airspace, and assume it is designated for arrival traffic. This arrival quadrant is our search space, which is essentially a 2-dimensional domain, and the merge points of the tree we are seeking could lie anywhere in the 2D continuum. However, we do not want to search for merge points in the whole continuum; we want to be able to discretize the 2D domain in a reasonable way so that there is only a finite number of candidate locations where merge points could be, and that the solution obtained from searching in the finite search space (more specifically, a search graph) approximates the solution from searching the whole continuum.

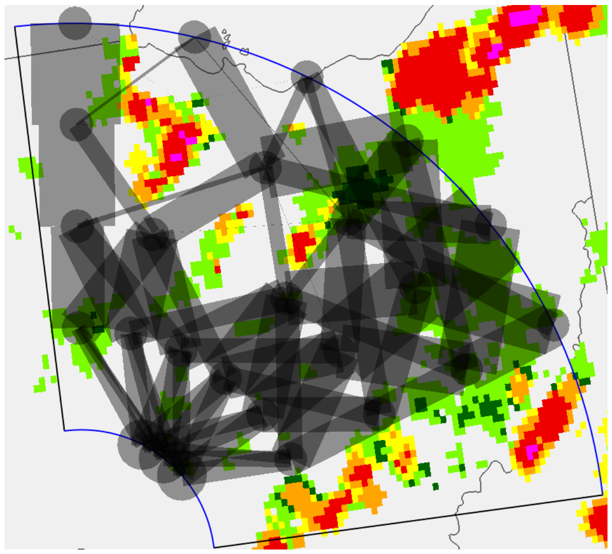


Figure 37: An example of a layered DAG computed using TBRP. Thickness of any edge is its maximum weather clearance value and is assigned to be the edge weight.

We consider there are n equally spaced (in the radial direction) $90°$ concentric arcs in the quadrant, any two of which are at least distance L apart along the radial direction. On each circular arc, we place k equally spaced points (vertices) and mark them as candidate merge point locations. We refer to each circular arc together with the candidate merge point locations on it as a layer, and denote all n layers using $\mathcal{L}_1$ to $\mathcal{L}_n$ from the outer range ring to the inner range ring. (In fact, given a weather snapshot, or equivalently a set of obstacles, we only need to consider those candidate locations which are at least distance r away from any obstacle.) The small gray disks in Figure 37 represent candidate merge points; note that only available merge point locations are shown. The vertices on the outermost layer, $\mathcal{L}_1$, serve as entry points for incoming traffic, and the vertices on the innermost layer, $\mathcal{L}_n$, are the metering fixes. We connect edges only between vertices on adjacent layers, but we do not connect all pairs of vertices from two adjacent layers because we do not want the tree paths to deviate too much from the radial direction, which is the desired the flight direction. So from any vertex v in layer $\mathcal{L}_i$, we connect a directed edge from v to any vertex u on $\mathcal{L}_{i+1}$ such that edge (u, v) is within a constant angular range from the radial direction; we then assign weights w_e to each edge e as described in the previous section. We also put an imaginary sink vertex t at the center of the annulus and connect all vertices on $\mathcal{L}_n$ to it with weights set to $+\infty$. The resulting search graph is a layered Directed Acyclic Graph (DAG) with weights, and it has k source vertices $s_1, \ldots, s_k$, $s_i \in \mathcal{L}_1$, and one sink vertex t. See Figure 37 for an example of the layered DAG computed using our TBRP system.

Problem Statement After obtaining a search DAG computed from a quadrant and hazardous weather as input, our problem is, given a demand profile of arrival flows, how to find a fan-in tree as a subgraph of the DAG that connect arrival entry points, $\mathcal{L}_1 = \{s_1, \ldots, s_k\}$, to the sink t while satisfying the maximum in-degree constraint Δ and RNP constraints, such that the tree accommodates as many demand as possible (or optimizes other objective functions).

6.2 Hardness Results

In this section, we consider the tree routing problem as a pure combinatorial optimization problem and give hardness results for two versions of the problem. Here we simplify the demand profile to one flight per entry point and ignore the RNP constraint, as the full specification of demand profile plus RNP constraint are essential to the complexity of the problem. The basic form of the problem we study in this section are thus, given a DAG with sources $\mathcal{L}_1 = \{s_1, \ldots, s_k\}$ and one sink t, to determine if there exists a in-degree bounded tree connecting $\mathcal{L}_1$ to t.

Related Work While we know of no prior work on finding bounded degree trees in directed acyclic graphs (DAG), there are abundant literature on finding vertex/edge-disjoint paths in DAGs. Vygen [52] showed that the directed edge-disjoint paths problem is NP-complete (the author actually proved more specialized cases to be NP-complete). Perl and Shiloach [44] studied the special case of finding pairs of disjoint paths in a DAG, $G = (V, A)$, connecting two given pairs of terminals, and they gave an algorithm in time $O(|V| \cdot |A|)$ for the vertex-disjoint case. Eppstein [14] also studied this problem of finding pairs of disjoint paths in a DAG and some other extensions.

Degree-bounded Tree in a DAG The input to our problem is a directed acyclic graph (DAG), $G(V, A)$, with one single *sink* t and multiple *sources* $s_1, s_2, \ldots, s_k$. One decision question is to determine if there is in-degree bounded tree servicing all the source vertices; we show this problem is NP-hard as stated in the following theorem:

Theorem 6.1. *Given a DAG $G = (V, A)$ with source vertices $\mathcal{L}_1 = \{s_1, \ldots, s_k\}$ and sink vertex t, it is NP-hard to decide if G contains a tree, $T = (V_T, A_T)$, as a subgraph such that $\mathcal{L}_1 \cup \{t\} \subset V_T$, and the maximum in-degree of T is bounded by a pre-specified number Δ.*

Proof. We reduce from 3-SATISFIABILITY (3SAT) [18]. For any 3SAT instance, let $U = \{u_1, u_2, \ldots, u_n\}$ be a set of variables and $C = \{c_1, c_2, \ldots, c_m\}$ be the set of clauses. We shall construct a geometric DAG, G, such that C is satisfiable if and only if G contains a subgraph satisfying all the three conditions.

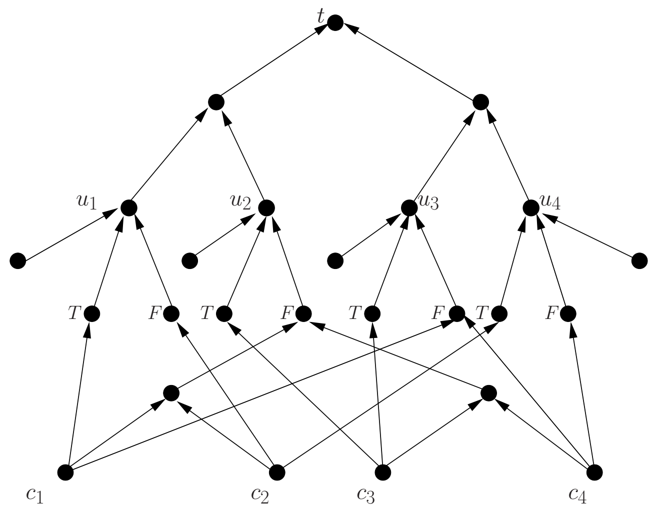


Figure 38: The graph G corresponding to a 3SAT instance of $(u_1 + \bar{u}_2 + \bar{u}_3)(\bar{u}_2 + \bar{u}_1 + u_4)(u_2 + u_3 + \bar{u}_2)(\bar{u}_2 + \bar{u}_3 + \bar{u}_4)$ with $\Delta = 2$

We create an internal vertex for each variable u_i. The sink vertex t uses an arbitrary in-degree Δ tree connecting to all the variables vertices. Each variable vertex u_i has $\Delta - 1$ designated source vertices that are only connected to it, so that it can only have one more incoming vertex due to the in-degree constraint. Directly connected to u_i are also two internal vertices designated as its truth literal and false literal, and only one of them may be connected to u_i in any feasible solution (refer to figure 38). We create a source vertex for each clause c_i. Each literal vertex builds an arbitrary in-degree Δ tree connecting to all the clauses vertices that it participates in. In this way, we have created a DAG G with $(\Delta - 1)n + m$ source vertices, and it is easy to see that the 3SAT instance, C, is satisfiable if and only if the decision problem for G returns yes.

□

Adding Simplicity Constraint We add one more constraint to the version of the problem considered above by requiring that the tree sought have a simple planar drawing, in order to better model the full tree routing problem we are solving; we show that this problem is also NP-hard as stated in the following theorem:

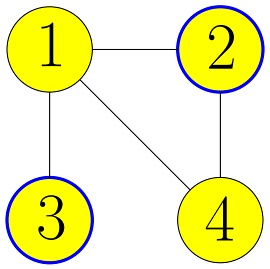
Theorem 6.2. *Given a drawing of a DAG $G = (V, A)$ with source vertices $\mathcal{L}_1 = \{s_1, s_2, \ldots, s_k\}$ and sink vertex t, it is NP-hard to decide if G contains a tree, $T = (V_T, A_T)$, as a subgraph such that $S \cup \{t\} \subset V_T$, the maximum in-degree of T is bounded by a pre-specified number Δ, and T does not have crossing edges (within the drawing of G).*

Proof. We use a reduction from INDEPENDENT SET [18]. For any given graph $G = (V, E)$ with $|V| = n$ and $|E| = m$ and any given integer $k \leq n$, we construct a DAG G', shown in Figure 39, such that there exists an independent set of size k in G if and only if our tree routing problem for G' returns yes.

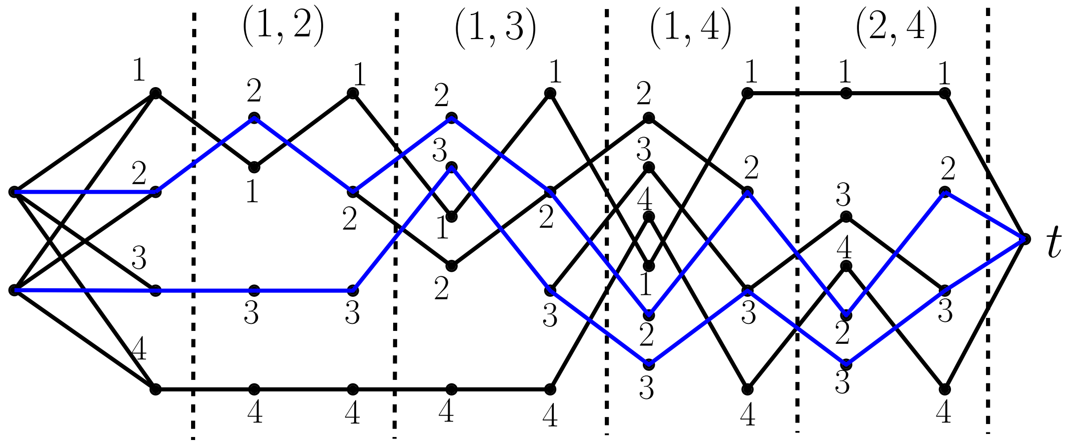
We layout G' in a layered graph drawing [8]. G' has a total of $2m + 2$ vertical layers plus a sink vertex t, with vertices of each layer aligned on the same (invisible) vertical line. Vertices on each layer (except for $\mathcal{L}_1$) gets a label from $\{1, \ldots, n\}$, which correspond to vertices in G, and edges are drawn between vertices from adjacent layers only if they share the same label. Our construction will have the property that the only tree that connecting all the $\mathcal{L}_1$ to t consisting of k non-crossing paths, each corresponding to a vertex in the desired independent set, that join each other only at the sink t; we will refer to the path corresponding to any vertex i as path π_i, etc.

We draw k vertices on the first layer, $\mathcal{L}_1$, to serve as the source vertices, $\mathcal{L}_1$, in the DAG. On the second layer, $\mathcal{L}_2$, we draw one vertex for each vertex $v \in G$, and they are labeled from 1 to n from top to bottom. we connect any vertex on $\mathcal{L}_1$ to any vertex on $\mathcal{L}_2$ to form a complete bipartite graph. These two layers form the vertex gadgets of our construction.

Each edge gadget takes up two layers and they are put next to each other in an arbitrary order as shown in Figure 39. For each edge (i, j) $(i < j)$ in



(a) An instance of INDEPENDENT SET



(b) The blue paths form a tree that corresponds to independent set $\{2,3\}$.

Figure 39: Hardness gadgets for a 4-vertex graph.

E, the second layer of its edge gadget has n vertices labeled from 1 to n from top to bottom; its first layer is labeled as $1, 2, \ldots, i-1, i+1, \ldots, j-1, j, i, i+1, \ldots, j-1, j+1, \ldots, n$ such that it only creates a crossing for path π_i and π_j, while allowing all the other paths to pass this gadget without crossing.

From our construction, it is easy to see that k non-crossing paths exist if only if they pass all the edge gadgets without incurring any crossing, i.e., no two of their corresponding vertices are adjacent in G. Therefore G' contains a tree connecting $\mathcal{L}_1$ to t without crossing edges if and only if G has an independent set of size k. □

6.3 Algorithm

In this section we consider the full tree routing problem as formulated in 6.1.2 and we follow the notations introduced in that section as well. We devise a dynamic programming algorithm to solve the tree routing problem exactly, in time exponential only to the number of layers n. For the convenience of presentation, here we focus on the description of the algorithm with respect to a specific objective function: that of maximizing the total number of arrival flights accommodated, although the algorithm is applicable to a wide range of other objective functions.

Again for simplicity of the presentation, we assume that the demand profile is encoded as a n-tuple, $(d_1, \ldots, d_k)$, $d_i \geq 0$, meaning that, for each i, there are d_i flights coming into entry point s_i along $\mathcal{L}_1$. We assume that once any entry point s_i is used by a tree, all the d_i flights can be accommodated by the tree using this entry point. Note that we have intentionally ignored the RNP constraint for each flight in the demand as we feel they are technical details only relevant to the implementation; in the implementation what we do is to consider the maximum RNP of all the flights coming into s_i as the uniform RNP for all of them, and the RNP requirements are always been checked against the edge weights of the paths we are searching. We also assume the in-degree bound, Δ, is equal to 2. Now our objective is, given a DAG, $G = (V, E)$, embedded in a quadrant, with sources $\mathcal{L}_1 = \{s_1, \ldots, s_k\}$ and sink t, and given a demand $(d_1, \ldots, d_k)$, find an in-degree 2 tree $T = (V_T, E_T)$ in G such that $t \in V_T$, there are no crossing edges, and the following objective function is optimized:

$$\sum_{s \in V_T \cap \mathcal{L}_1} d_s.$$

An example of the optimal tree computed from the DAG in Figure 37 using all RNP-1 demand on all entry points is given in Figure 40.

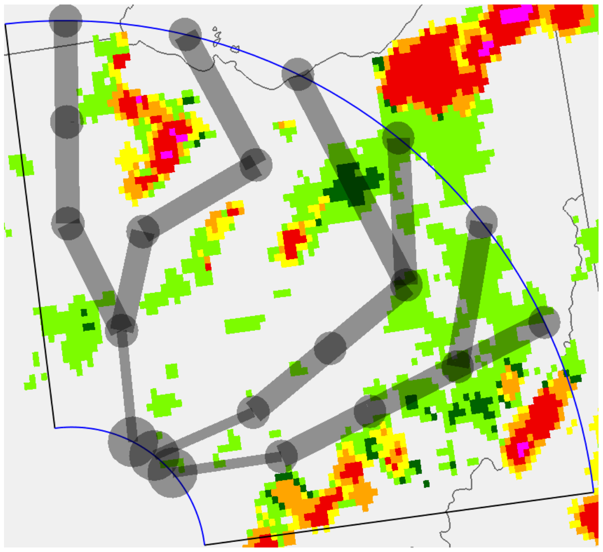


Figure 40: An example of an optimal tree computed using TBRP. Thickness of any edge correspond to the RNP values of the demand flights; they are all of RNP-1 in this example.

State of the Dynamic Algorithm Suppose T is a solution to our problem. Note that T are not allowed to have any crossing edges. Consider an arbitrary path π connecting any source vertex s_i to the sink R. Imagine if we delete π from T, the parts left on both sides of π are collections of subtrees of T, with each subtree attached to the "backbone" π in T; refer to Figure 41. Following this observation, our dynamic programming basically works from one side of the DAG to the other, connecting new path to the current tree into a new tree, and recursing into the sub-problem defined as the part of the DAG to the "left" of the new path.

We assume there is an orientation along the circular direction (one that is orthogonal to the radial direction) so that we can speak of the "left" and the "right" of any path that goes all the way to t. We assume the algorithm works from right to left in the DAG. For any entry point s_i that has a positive demand on it, we say it has been "serviced" by the tree if the current tree has included s_i, we say it has been ignored if the current tree decides not to include it and proceeds to the left of it, and we refer to all the other ones as entry points that still "need service". A sub-problem of the dynamic programming

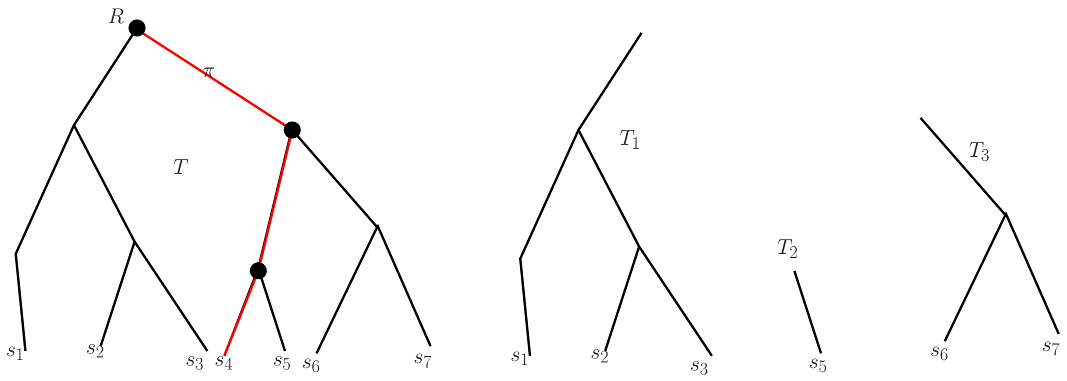


Figure 41: T is one solution to our problem servicing entry points $s_1, \ldots, s_7$. After removal of a tree-path π, T is decomposed into three subtrees T_1, T_2, T_3, each servicing a consecutive range of entry points. The black dots are showing attachment vertices of T_1, T_2, T_3 to π.

then consists of a entry point s_i indicating all entry points to the left of it still need service, a source to sink path π separating the "done" and "undone" parts of the dynamic programming so that the part of G to the left of π are available, and an available in-degree sequence, $\Delta_\pi = (\Delta_1, \ldots, \Delta_n)$, $\Delta_i \leq \Delta$, of π, indicating for each point on π, the number of new incoming edges it can have so as not to violate the overall in-degree constraint Δ. For any sub-problem defined by (s_i, π, Δ_π), the state of the dynamic programming, $f(s_i, \pi, \Delta_\pi)$, is thus defined as the maximum number of flights that can be accommodated by any collection trees within the subproblem such that each tree is connected to π and there are no crossing edges.

State Transition For any state $f(s_i, \pi, \Delta_\pi)$, we will first try to service s_i by considering all possible paths from s_i to t either directly or via π, trying each new path π', and solving the new sub-problem $(s_j, \pi', \Delta_{\pi'})$ where s_j is the next entry point along the "leftward direction" that wants to be serviced, and $\Delta_{\pi'}$ is the updated available in-degree sequence for π'; we then consider the option of the ignoring s_j and compute the sub-problem, (s_j, π, Δ_π). We will pick from the states of all the sub-problems the one that is the maximum and record it as the "contributing state" of state $f(s_i, \pi, \Delta_\pi)$; we will use this information to construct an optimal tree when we finish computing all the states. If we denote by G_π the part of the DAG G to the left of π and including π but with the in-degree constraints Δ_π, the state transition equation can be written as

$$f(s_i, \pi, \Delta_\pi) = \max \left\{ d_{s_i} + \max_{\pi' \in G_\pi} f(s_j, \pi', \Delta_{\pi'}), f(s_j, \pi, \Delta_\pi) \right\}$$

6.4 Experiments

We have implemented the dynamic algorithm in a software system called *Tree-based Route Planner* (TBRP). We design several experiments using TBRP in order to demonstrate our capability to find optimal trees under different objective functions, to handle both static and dynamic metering fix requirements, with different weather systems, and considering both constant and mixed RNP requirements.

Experiment Setup We carried out several experiments comparing the results of our software (TBRP) running under different parameters. The set of parameters that is common to all experiments are:

Weather timestamp: 2005/06/28 21:48:00 Zulu unless specified otherwise

Quadrant: centered at latitude 39.98° N, longitude 84.50° W, with inner and outer radii of 40 and 150 nmi

Demand Profile: one RNP-2 flight per arrival entry arc, unless specified otherwise

NWS Threshold: level 3 and above (colored as yellow or more severe) is considered as obstacle

Metering Fixes: dynamic metering fix and 3X, unless specified otherwise

Entry Arcs: seven segments on each 90° quadrant

Layers: four layers for merge points (plus one layer for metering fixes)

Objective Functions For any tree $T = (V_T, E_T)$ in the given DAG G and any of its entry point $s \in \mathcal{L}_1$, we let π_s be the unique path from s to the sink t within T and call it a tree path. An objective function, f, is a function that maps the set of such trees to a totally ordered set. We want to find, for some reasonable choice of objective function f, a tree T^* such that $f(T^*)$ is optimized. Algorithmically, our dynamic programming algorithm is capable of finding optimal trees for any objective function that can be constructed recursively on the set of tree paths. For simplicity of implementation, our current code optimizes two objective functions. For any given tree $T = (V_T, E_T)$, we define two metrics on T :

$$f(T) = \sum_{s \in V_T \cap \mathcal{L}_1} d_s$$

measuring the total demand that can be accommodated by T, and

$$g(T) = \sum_{s \in V_T \cap \mathcal{L}_1} d_s \cdot L(\pi_s)$$

where $L(\pi_s)$ is the arc length of tree path π_v, measuring the total flight distance from entry point to metering fix by all the demand accommodated.

Our first objective function is simply $f(T)$. Our second objective function is $h(T) = (f(T), -g(T))$, and we maximize it under lexicographical order, that is, we want to first maximize the total demand, $f(T)$, and for all trees achieving the same maximum demand, we then minimize the total flight distance, $g(T)$. Unless specified otherwise, all of our experiments were run under the more optimized objective $h(T)$.

Demand-maximized versus Demand-length-optimized Trees Any optimal tree found for the objective $f(T)$ is called a demand-maximized tree, and any optimal tree for the objective function $h(T)$ a called a demand-length-optimized tree. We run experiments on the same input instance with the two objective functions, and for each optimal tree found, we record both the metric $f(T)$ and $g(T)$. We see in Figure 42 that $f(T) = 18$ in both cases, which is exactly the number of tree branches. And the demand-length-optimized tree has about 4% less total flight distance than the demand-maximized tree.

Static versus Dynamic Metering Fixes For a quadrant centered at latitude 40.58° N, longitude 83.90° W, we run the experiment under both static and dynamic metering fix options, as illustrated in Figure 43. For static metering fixes, the locations of the metering fixes are fixed so if any metering fix is blocked by weather, it is considered infeasible. The location for the center fix is at the angular center of the quadrant, 40 nmi from the center of the metroplex. The locations for the other two are exactly distance d away (d is a parameter equal to 7.5 nmi in all the experiments here).

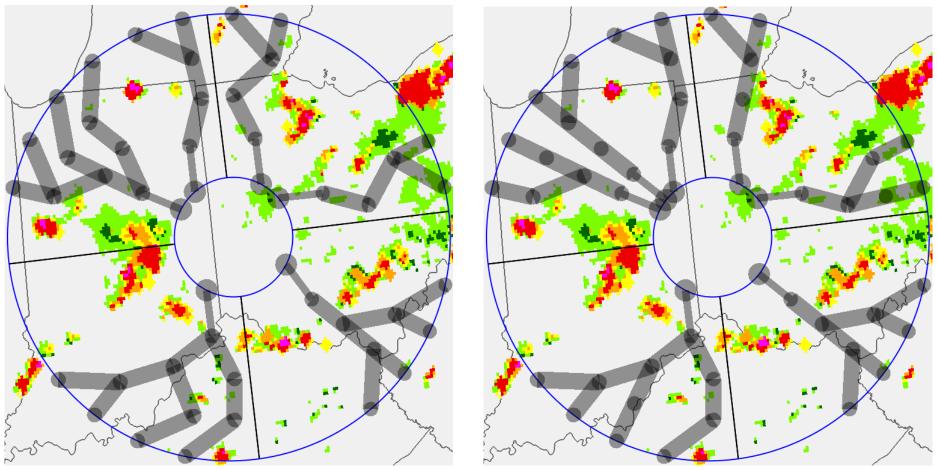

(a) f(T) = 18 flights, g(T) = 3045.13 nmi (b) f(T) = 18 flights, g(T) =2923 nmi

Figure 42: Demand-maximized versus demand-length-optimized trees.

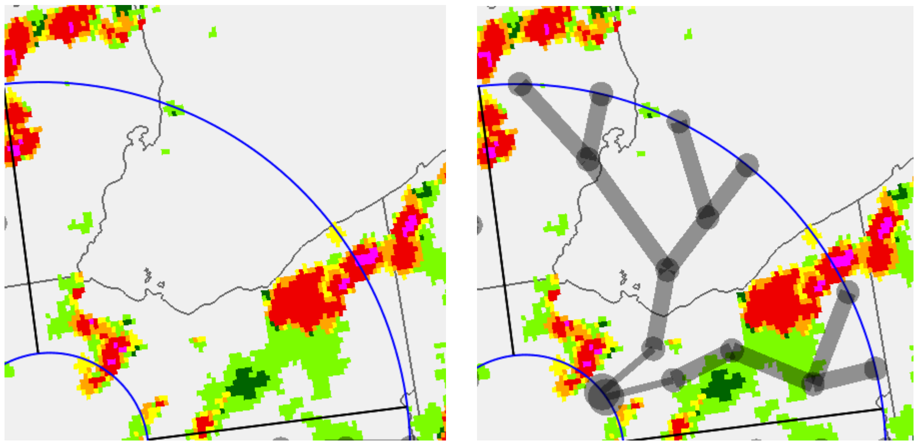

(a) (b)

Figure 43: Trees with static and dynamic metering fixes.

For dynamic metering fixes, we optimize the locations of three metering fixes so that the number of metering fixes possible is maximized and stay as close as possible to the default locations for the static option. We use the following algorithms to determine the locations of metering fixes:

Dynamic Metering Fix Generation We developed an algorithm for dynamically generating weather-free metering fixes in order to permit us to model the dynamic metering fix case.

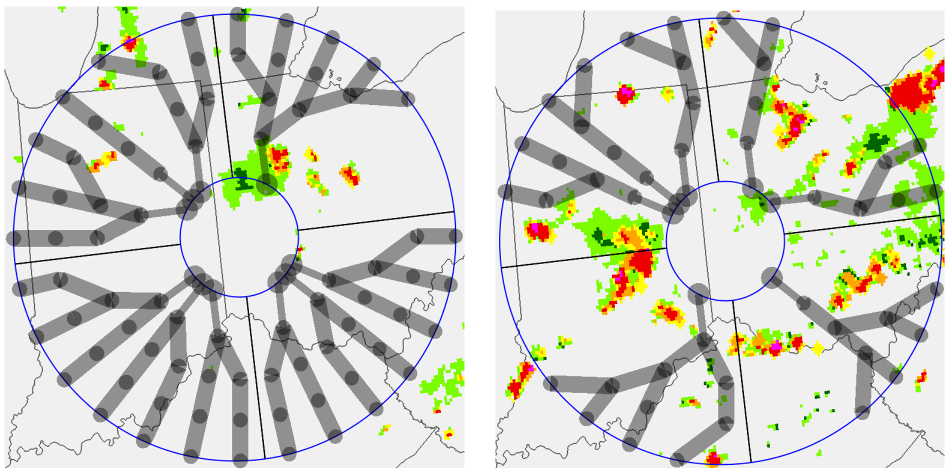
Metering fixes are modeled as points along a circular arc of 40 nmi radius around the center of the metroplex. We use a minimum separation requirement, δ, between any two metering fixes, that is RNP-independent; every metering fix location must be clear of hazardous weather and other metering fixes to a distance of δ nmi, effectively creating a circle of exclusion in which the airspace must be clear. (Another metering fix may be located immediately outside of this circle.) For NextGen, we want to find, ideally, three metering fixes in each quadrant satisfying the above constraints, and are located as close as possible to the original, default location of the metering fixes. (The default locations are inputs to the algorithm.)

For purposes of our experiments, we assumed default locations for the metering fixes as follows. The center metering fix is at the angular center of each quadrant, 40 nmi from the center of the metroplex. For example, in the northeast quadrant, the center metering fix is at the 45° compass heading. The other two default metering fixes are exactly distance δ away from the center fix on either side. These default metering fixes were used as the fixed metering fix locations for all experiments, and used as the default metering fix locations for the dynamic metering fix generation algorithm and experiments.

In order to calculate dynamic metering fixes, we first determine a location for the center fix as close to its corresponding default location as possible. We start from the default position and iterate through other possible center fix points according to their heading, such that each new point represents a change of heading of ε degrees. We start from the smallest possible change and search the points in order of increasing change. For example, in the northwest quadrant, we would check points at different headings in this order: $(45, 45+\varepsilon, 45-\varepsilon, 45+2\varepsilon, 45-2\varepsilon, \text{etc.})$. The first point that is a feasible location is chosen as the center fix. From this center fix location, we pick headings that are exactly δ distance away on both sides. From there, we move further and further away in increments of ε degrees of heading until we find suitable points on both sides.

If we do not find the maximum possible number of metering fixes because our choice of the center fix makes it impossible, then this is because we only found one or two fixes. If we found two, then we move our choice of center fix to the other fix, and attempt to search for fixes once again. If we only found one, then we stop; there is only one fix available under the current level of discretization (controlled by ε).

Light versus Moderate Weather In this experiment, we compare light and moderate weather conditions. The light weather is taken at timestamp 2005/06/28 14:01:00 Zulu and the moderate one is again at 2005/06/28 21:48:00 Zulu. The demand profile to both cases is pure RNP-2 as specified in the experiment setup section. We see that, as illustrated in Figure 44 under the same demand profile, we can accommodate about 39% more incoming traffic in the light weather than in the moderate weather.

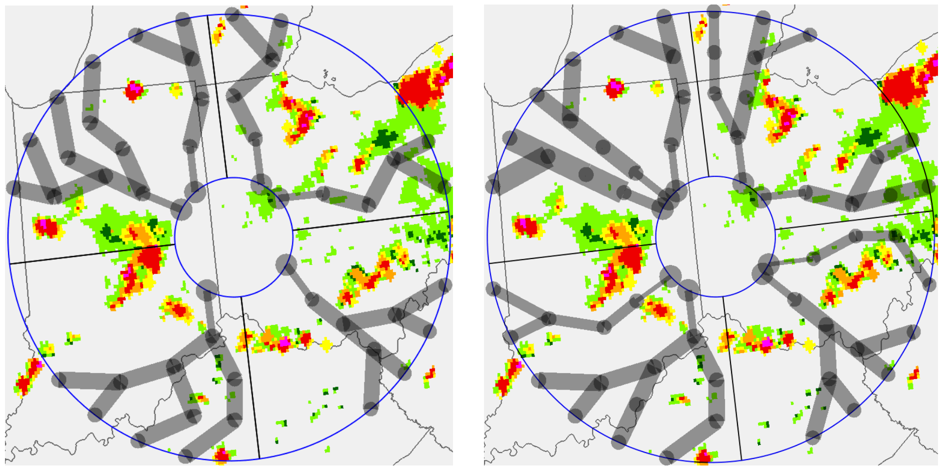


(a) $f(T) = 25$ flights (b) $f(T) = 18$ flights

Figure 44: Trees under light and moderate weather.

Constant versus Mixed RNP This experiment considers two different demand profiles: one is the constant RNP-2 profile, and the other one is similar in the sense that there is only one incoming flight at each arrival arc but they come with different RNP requirements (ranging from RNP-1, RNP-2, to RNP-3), hand-picked in order to better utilize the airspace. (If the order of the RNPs is not given, this problem becomes NP-Hard, as shown in [24]

The results shown in Figure 45 demonstrate that, with our programs mixed RNP capability, we can accommodate more flights with wider range of RNP capabilities.



(a) 18 RNP-2 flights

(b) 22 flights (5 RNP-1, 15 RNP-2, and 2 RNP-3)

Figure 45: Constant RNP-2 tree versus Mixed RNP tree

6.5 Conclusion

This work presents a theoretical method and experimental results for flexible tree-based planning in transitional airspace. The basic version of the problem, deciding the existence of a non-crossing tree in a given DAG, turns out to be NP-hard. We presented an algorithm to solve the problem exactly based on dynamic programming and implemented the algorithm in our TBRP system. We demonstrated in the experiments our system's ability to optimize over different objective functions, to handle mixed RNP requirement, and to compute dynamic metering fixes.

The algorithm we presented takes a snapshot of weather as input and computes the best weather avoidance tree; this is in a sense a "static" algorithm. However when we consider dynamic weather, we hope to be able to avoid recomputing an entirely new solution whenever weather moves or changes, and to re-use as much of the existing solution as possible when accommodating presumably small changes in weather from every time stamp to the next.

References

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

[2] N. Amato. Determining the separation of simple polygons. *International Journal of Computational Geometry and Applications*, 4(4):457–474, 1994.

[3] E. M. Arkin, J. S. B. Mitchell, and V. Polishchuk. Maximum thick paths in static and dynamic environments. In *Proceedings 24th ACM Symposium on Computational Geometry*, pages 20–27, 2008.

[4] D. Bertsimas and S. Patterson. The air traffic flow management problem with en route capacities. *Operations Research*, 46:406–422, May–June 1998.

[5] D. Bertsimas and S. Patterson. The traffic flow management rerouting problem in air traffic control: A dynamic network flow approach. *Transportation Science*, 34(3):239–255, 2000.

[6] P. Bose, L. Devroye, M. Löffler, J. Snoeyink, and V. Verma. The spanning ratio of the delaunay triangulation is greater than $pi/2$. In *Proceedings of the 21st Canadian Conference on Computational Geometry (CCCG2009)*, pages 165–167, 2009.

[7] P. Bose, A. Maheshwari, G. Narasimhan, M. Smid, and N. Zeh. Approximating geometric bottleneck shortest paths. *Computational Geometry*, 29(3):233–249, Nov. 2004.

[8] M. Carpano. Automatic display of hierarchized graphs for computer-aided decision analysis. *IEEE Transactions on Systems, Man and Cybernetics*, 10(11):705–715, 1980.

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

[10] F. Y. Chin, J. Snoeyink, and C. A. Wang. Finding the medial axis of a simple polygon in linear time. In *ISAAC '95: Proceedings of the 6th International Symposium on Algorithms and Computation*, pages 382–391, London, UK, 1995. Springer-Verlag.

[11] M. Dixon and G. Weiner. Automated aircraft routing through weather-impacted airspace. In *5th International Conference on Aviation Weather Systems*, pages 295–298, Vienna, VA, 1993.

[12] D. P. Dobkin, S. J. Friedman, and K. J. Supowit. Delaunay graphs are almost as good as complete graphs. *Discrete Comput. Geom.*, 5(4):399–407, 1990.

[13] H. Edelsbrunner, L. J. Guibas, and J. Stolfi. Optimal point location in a monotone subdivision. *SIAM J. Comput.*, 15(2):317–340, 1986.

[14] D. Eppstein. Finding common ancestors and disjoint paths in DAGs. Report 95-52, University of California, Irvine, Dec. 1995.

[15] J. Evans and E. Ducot. The integrated terminal weather system (ITWS). *The Lincoln Laboratory Journal*, 7(2):449–474, Fall 1994.

[16] FAA. 2003 aviation capacity enhancement plan. *Federal Aviation Administration Office of System Capacity*, 2003.

[17] M. Farshi. *A Theoretical and Experimental Study of Geometric Networks*. PhD thesis, Eindhoven University of Technology, 2008.

[18] 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.

[19] L. Gewali, A. Meng, J. S. Mitchell, and S. Ntafos. Path planning in $0/1/\infty$ weighted regions with applications. In *SCG '88: Proceedings of the fourth annual symposium on Computational geometry*, pages 266–278, New York, NY, USA, 1988. ACM.

[20] M. Held. Vroni: An engineering approach to the reliable and efficient computation of voronoi diagrams of points and line segments. *Comput. Geom.*, 18(2):95–123, 2001.

[21] Joint Planning and Development Office. Next generation air transportation system: Weather concept of operations. Version 1.0, Washington, DC, 2006.

[22] J. M. Keil and C. A. Gutwin. Classes of graphs which approximate the complete euclidean graph. *Discrete Comput. Geom.*, 7(1):13–28, 1992.

[23] J. Kim, J. S. Mitchell, and J. Zou. Approximating maximum flow in polygonal domains using spanners. In *Proceedings of the 21st Canadian Conference on Computational Geometry (CCCG2009)*, pages 115–118, 2009.

[24] J. Kim, J. S. B. Mitchell, V. Polishchuk, S. Yang, and J. Zou. Routing multi-class traffic flows in the plane. Technical report, Stony Brook University, 2009. unpublished draft.

[25] J. Krozel, B. Capozzi, T. Andre, and P. Smith. The future national airspace system: Design requirements imposed by weather constraints. In *AIAA Guidance, Navigation, and Control Conf.*, Austin, TX, Aug 2003.

[26] J. Krozel, B. Hoffman, S. Penny, and T. Butler. Aggregate statistics of the national airspace system. In *AIAA Guidance, Navigation, and Control Conf.*, Austin, TX, Aug 2003.

[27] J. Krozel, C. Lee, and J. Mitchell. Estimating time of arrival in heavy weather conditions. In *AIAA Guidance, Navigation, and Control Conf.*, Portland, OR, Aug 1999.

[28] J. Krozel, J. 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.

[29] 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.

[30] 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.*, Aug 2008.

[31] J. Krozel, T. Weidner, and G. Hunter. Terminal area guidance incorporating heavy weather. In *AIAA Guidance, Navigation, and Control Conf.*, pages 411–421, New Orleans, LA, Aug 1997.

[32] M. Lanthier, A. Maheshwari, and J.-R. Sack. Approximating weighted shortest paths on polyhedral surfaces. In *Proceedings 13th ACM Symposium on Computational Geometry*, pages 485–486, New York, NY, USA, 1997. ACM.

[33] C. S. Mata and J. S. B. Mitchell. A new algorithm for computing shortest paths in weighted planar subdivisions (extended abstract). In *Proceedings 13th ACM Symposium on Computational Geometry*, pages 264–273, New York, NY, USA, 1997. ACM.

[34] J. S. B. Mitchell. On maximum flows in polyhedral domains. *J. Comput. Syst. Sci.*, 40:88–123, 1990.

[35] J. S. B. Mitchell and C. H. Papadimitriou. The weighted region problem: finding shortest paths through a weighted planar subdivision. *J. ACM*, 38(1):18–73, 1991.

[36] J. S. B. Mitchell and V. Polishchuk. Thick non-crossing paths and minimum-cost flows in polygonal domains. In *Proceedings 23rd ACM Symposium on Computational Geometry*, pages 56–65, 2007.

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

[38] V. Polishchuk. *Thick non-crossing paths and minimum-cost continuous flows in polygonal domains*. PhD thesis, Stony Brook University, 2007.

[39] J. Prete. *Aircraft Routing in the Presence of Hazardous Weather*. PhD thesis, Stony Brook University, Aug 2007.

[40] J. Prete and J. Mitchell. Safe routing of multiple aircraft flows in the presence of time-varying weather data. In *AIAA Guidance, Navigation, and Control Conf*, 2004.

[41] R. Richbourg, N. Rowe, M. Zyda, and R. McGhee. Solving global two-dimensional routing problems using snell's law and a search. In *1987 IEEE International Conference on Robotics and Automation. Proceedings*, volume 4, 1987.

[42] RTCA. Report of the rtca board of directors' select committee on free flight. Technical report, Wash., DC, Jan 1995.

[43] D. Schmidt. On modeling ATC work load and sector capacity. *Journal of Aircraft*, 13(7):531–537, 1976.

[44] Y. Shiloach and Y. Perl. Finding two disjoint paths between two pairs of vertices in a graph. *J. ACM*, 25(1):1–9, 1978.

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

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

[47] 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.

[48] M. Steiner, R. Bateman, D. Meganhardt, , and J. Pinto. Evaluation of ensemble-based probabilistic weather information for air traffic management. In *Aviation, Range, and Aerospace Meteorology, Special Symposium on Weather — Air Traffic Management Integration*, Phoenix, AZ, Jan 2009.

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

[50] 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.

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

[52] J. Vygen. Np-completeness of some edge-disjoint paths problems. *Discrete Appl. Math.*, 61(1):83–90, 1995.

[53] C. Wanke, L. Song, S. Zobell, D. Greenbaum, and S. Mulgund. Probabilistic congestion management. In *6 th USA/Europe Seminar of Air Traffic Management R&D*, pages 27–30, 2005.

[54] A. Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM J. Comput.*, 11(4):721–736, 1982.

[55] J. Zou, J. Kim, J. W. Krozel, J. Krozel, and J. S. B. Mitchell. Two methods for computing directional capacity given convective weather constraints. In *AIAA Guidance, Navigation, and Control Conf.*, Aug 2009.