Stony Brook University

Geometric Algorithms for Dynamic Airspace Sectorization

A Dissertation Presented
by
Girishkumar R. Sabhnani
to
The Graduate School
in Partial fulfillment of the
Requirements
for the Degree of

Doctor of Philosophy
in
Computer Science

Stony Brook University
May 2009

Stony Brook University
The Graduate School

Girishkumar R. Sabhnani

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

Joseph S. B. Mitchell - Dissertation Advisor
Professor, Computer Science Department

Esther M. Arkin - Chairperson of Defense
Professor, Computer Science Department

Jie Gao - Committee Member
Assistant Professor, Computer Science Department

George Hart - Committee Member
Research Professor, Computer Science Department

Dr. Robert L. Hoffman - Outside Member
PhD Applied Mathematics, Metron Aviation, Inc. and Institute for Systems Research at the University of Maryland, College Park

This dissertation is accepted by the Graduate School

Lawrence Martin
Dean of the Graduate School

Abstract of the Dissertation
Geometric Algorithms for Dynamic Airspace Sectorization
by
Girishkumar R. Sabhnani

Doctor of Philosophy
in
Computer Science

Stony Brook University
2009

The National Airspace System (NAS) is designed to accommodate a large number of flights over North America. For purposes of workload limitations for air traffic controllers, the airspace is partitioned into approximately 600 sectors; each sector is observed by one or more controllers. In order to satisfy workload limitations for controllers, it is important that sectors be designed carefully according to the traffic patterns of flights, so that no sector becomes overloaded.

We formulate and study the airspace sectorization problem from an algorithmic point of view, modeling the problem of optimal sectorization as a geometric partition problem with constraints. We evaluate our algorithms experimentally. We conduct experiments using actual historical flight track data for the NAS as the basis of our partitioning. We compare the workload balance of our methods to that of the existing set of sectors for the NAS and find that our resectorization yields competitive and improved workload balancing. In particular, our methods yield an improvement by a factor between 2 and 3 over the current sectorization in terms of the time-average and the worst-case workloads of the maximum workload sector.

Further, we investigate the dynamic nature of air traffic and use that to guide sector designs that evolve over time. Depending on the time of day, demand profiles, weather changes, etc. the traffic density of various parts of the NAS changes. In such a scenario, it is more practical to have dynamic sector designs in order to accommodate the changing traffic; in fact this is a common practice even today. The goal is to automate the identification and re-configuration of these dense traffic areas. A simple solution would just compute separate sectorizations for different instances of air traffic within a

sliding time window. While this method gives excellent workload balance for each time window, it does not guarantee that the change in sector design is minimal and local to dense traffic regions; a feature which is very important for dynamic sectorization. Hence, we propose an approach which involves local merging and re-partitioning of neighboring sectors in high traffic density regions.

Contents

List of Figures

List of Tables

Acknowledgements

I begin by acknowledging the two people without whom I would not have pursued doctoral studies in the first place; my advisor Prof. Joseph S. B. Mitchell and my friend and fellow graduate student Amitabh Basu. I thank Joe for introducing me to Computational Geometry (CG), the topic that attracted me the most at Stony Brook University (SBU), and for welcoming me to his research group. His precious guidance and extensive support have made this dissertation possible. Joe is a keen problem solver, an excellent teacher, ingenious researcher and the most humble person I have ever met.

Further, I thank Prof. Jie Gao for my first hands on experience in conducting algorithmic research. It was a thrilling experience to witness her transform my sensor networks course project into a research publication. I owe a lot to Prof. Esther M. Arkin, whose keen insight and fertile ideas always helped me through times when I was stuck on a problem in my research. I also cherish the time I spent working in the algorithms lab under Prof. Steven Skiena. My instructors (especially Joe, Estie, Steve, Prof. Yair Tauman and Prof. Michael Bender) at SBU and the courses they taught were equally instrumental in igniting my interest in research.

My work benefited immensely from productive discussions with Dr. Robert Hoffman, Dr. Jimmy Krozel, Dr. Arash Yousefi, Bert Hackney and Dr. Joseph Prete of Metron Aviation, Inc. and Gregory L. Wong of NASA, Ames. Many of the problems investigated in this thesis were triggered by the ideas exchanged with them over teleconferences. Their domain expertise with Air Traffic Management allured me towards and enhanced my knowledge about the field.

I am grateful to my colleagues working in the CG group at SBU, in particular Jason Zou, Joondong Kim, Rik Sarkar, Eli Packer and Irina Kostitsyna. The blackboard discussions with them were crucial in much of the problem-solving process during my PhD. My development as a researcher gained a lot from my participation in the Reading Group and the CG Group meetings at Stony Brook. The workshops and conferences that I attended during my years as a PhD student were extremely useful. I specially profited from my visit to University of Helsinki and my host Valentin Polishchuk.

I value the research collaborations with Prof. Alon Efrat, Arizona State University and Dr. Alexander Kröller, Braunschweig Institute of Technology.

I thank Prof. Alan Tucker for giving me the opportunity to teach Probability Theory. I also acknowledge the help from other staff members at SBU, including Cynthia Scalzo, Christine Rota, Victor Poon, Brian Tria, Christos Kalesis and Dr. Kent Marks.

Last, but not the least, I thank my parents, brother, sister-in-law and friends whose constant encouragement was invaluable to me. In particular, I thank Mahesh for putting my interests ahead of his, Abhishek for his never ending belief in me and Rajul for always being there. I would like to dedicate this thesis to my father, whose unceasing efforts during my formative years have instilled the values which laid the foundation of this dissertation.

This research was partially supported by grants from the National Science Foundation (CCF-0528209, CCF-0729019) and the NextGen-Airspace Project of NASA's Airspace Systems Program (Order No. NNA07BB33C). Special thanks to Gregory L. Wong, Shannon Zelinski, Harry Swenson, and Dr. Parimal Kopardekar at NASA Ames Research Center for their support and insightful comments.

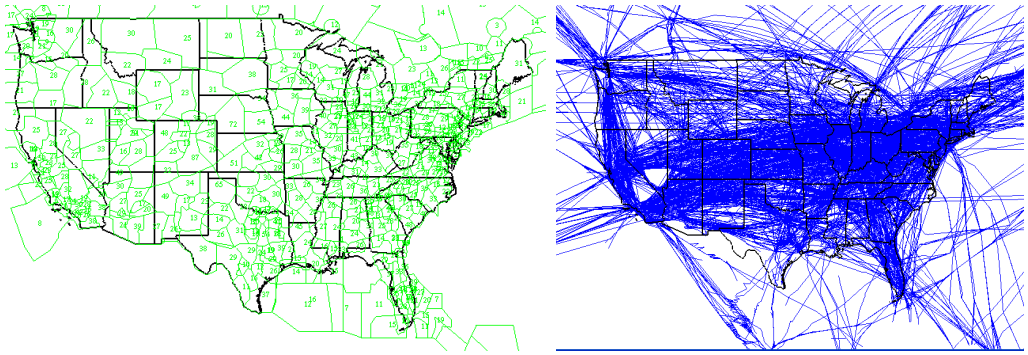
1 Introduction

1.1 Motivation

The National Airspace System (NAS) of United States (US) is a complex transportation system designed to facilitate the management of air traffic with safety as the primary objective and efficiency as the secondary objective. Airspace design engineers and air transportation policy makers are continually "tweaking" the system to adjust for changes in the demand patterns, changes in weather systems that disrupt the network, and changes in the air traffic management (ATM) policies that govern the safe operation of aircraft.

A key component of the NAS is the partitioning of airspace into managerial units. At the highest level, the NAS is partitioned into 20 Air Route Traffic Control Centers (ARTCC) (in the continental US), each of which is partitioned into *sectors*, each one of which is managed by one air traffic controller (ATC) (or a small team of 1-4 controllers) at any given time of the day. There are a total of about 600 sectors; the FAA employs about $15,000$ controllers, of which over 7000 are due to retire within the next 9 years [2], suggesting a need to redesign the airspace for fewer controllers in the near future. There are roughly $60,000$ daily flights within the NAS, interconnecting about 2000 airports (see Figure 1).

The capacity of the NAS to accommodate increases in traffic demand are being pushed to the limits. Both the FAA and NASA are backing initiatives to study how greater throughput can be accommodated safely through system redesign and new technologies for automation, communication, and ATM. The National Airspace Redesign (NAR) initiative [1] has been in place for the last few years to address this challenging problem. Airspace redesign is critical for anticipated future growth in the NAS. Current sector boundaries are largely determined by historical effects and have evolved over time. Hence, the sectors are too *rigid* i.e. the sector geometry has stayed relatively constant when compared to the change in the route structures and demand profiles over the years; and there is a large workload *imbalance* across the sectors.



(a) The current sectorization of the airspace. (b) Historical track data for flights.

Figure 1: (Part of) the National Airspace System of United States.

We study the automatic *sectorization* ("sector boundary design") of airspace problem from a formal and geometric perspective, while attempting to model precisely the system design constraints. In doing so, we have developed a tool, GeoSect, which allows us to explore algorithms and heuristics for automatic sectorization and load balancing. Load balancing here refers to balancing the workload of a controller managing the sector. The definition of workload is critical. It needs to take into account human factors issues, which include subjective estimations of psychological/physiological state and mental workload, such as issues of visual and auditory perception, memory, stress, and attention span. Many research studies (see, e.g., [32, 43, 5, 49]) have addressed the modeling and quantification of ATC workload.

We model the problem using a geometric and easily quantified approach to defining sector workload: Based on a given set of historical (or any other wind optimized, simulated etc.) flight data, $w(\sigma)$ is defined to be the maximum (worst-case) or the time average number of aircraft in sector σ during a fixed time window $[0, T]$ (typically, the time window corresponds to a 24-hour day). This definition accounts directly for the traffic density/number of flights aspect of workload. While it does not include other components that often make up an aggregated workload estimate, we are able to quantify some of these other factors and add them to our model. We have already done so for coordination workload between sectors (which accounts for the number of times a flight must be "handed off" between controllers), as we report later; other workload components for potential inclusion in our analysis include traffic mix, separa-

tion standards, aircraft speeds, crossing aircraft profiles, angle of intersection between routes, directions of flights, number of facilities in a sector, location of conflicts within a sector, number of altitude changes, etc (see [57] for more details).

The track data is assumed to be given. It gives a set of trajectories (each given by a sequence of *way points* with time stamps) for each recorded flight path in the NAS over the time window $[0, T]$. We are using the historical data to give a distribution (in space and time) of the typical trajectories of the aircraft in the NAS; on any given day, of course, the flight paths vary, with weather conditions and other events that disrupt the standard schedule. Thus, a potentially more desirable method of assessing workload is to use track data from an airspace simulation (such as NASA's Airspace Concept Evaluation System [53]), since this allows one to evaluate the "ideal" routes for a given set of demand, to incorporate new air traffic concepts (such as "Free Flight"), and to modify the demand according to predicted future growth. The methods we investigate, though, work equally well with input from a simulator or from historical data.

We further extend our methods to account for no-fly constraints and the location of airports within sectors. In accordance with controller feedbacks to our methods we enhance capabilities of GeoSect to have sector boundaries conform to dominant flows in air-traffic. The software also has capabilities of $3D$ sectorization in which each sector along with its $2D$ polygonal region, also has an altitude range, usually referred as *floor* (for low altitude limit) and *ceiling* (for high altitude limit).

After extensive development of methods for static sectorization (sector boundaries remains same over the day or entire season), we investigate methods to automate dynamic sectorization (sector boundaries evolve over the course of the day) to accommodate changes in traffic pattern due to changing demand profiles and the effect of convective weather over the NAS.

1.2 Related Work

The sectorization problem has been previously studied as a global optimization problem using techniques of integer programming; after discretizing the NAS into 2566 hexagonal cells, Yousefi and Donohue [60, 59] formulate and solve an extensive mathematical programming model that captures more of

the sector workload issues than many prior methods. They use a large-scale simulation to compute en route metrics that are combined to give a workload model. Yousefi's thesis [59] is a good reference for problem motivation and related literature prior to year 2004. More recent efforts in sectorization are summarized next. Bloem et al. [17] combine under-utilized airspace sectors to conserve air traffic control resources, Xue [58] starts with a Voronoi partition of 'k' points (if 'k' sectors are desired), and uses genetic algorithms to move the points so as to optimize workload balance, Conker et al. [22] also explore a clustering approach starting with square grid cells to design sector boundaries, but they explicitly refine sector boundaries to conform to traffic pattern. Lee's thesis [39] is a good recent resource for understanding airspace complexity.

In the algorithms literature, there has been related work on partitioning of rectangles and arrays for load balancing of processors; see, e.g., [14, 24, 15, 34, 35, 44, 45]. Geographical load balancing applications have arisen in political districting (to avoid gerrymandering); see Altman [6] (who proves NP-hardness of political districting), Altman and McDonald [7], and Forman and Yue [28]. Geographic load balancing also arises in electric power districting; see, e.g., Bergey, Ragsdale, and Hoskote [13]. Recent work in the computational geometry literature looks at minimum-cost load balancing in sensor networks; see Carmi and Katz [9].

What makes our sectorization problem novel compared with most geometric load balancing problems previously studied is that the input data consists of *trajectories* of *moving* points; typical geometric partitioning problems have addressed *static* point data. Also, dynamic aspect of the problem rises interesting questions, less studied in the literature.

1.3 Summary of Contributions

1. We model the airspace sectorization problem in algorithmic terms, as a precise computational geometric formulation.

2. We develop a suite of heuristics (extending exact solution to a 1D problem) to solve the problem in two dimensions (2D), and we discuss algorithmic issues.

3. We enhance the methods to incorporate other no-fly zone and special dominant flow crossing constraints.

4. We implement and conduct experiments to test the effectiveness of our methods on real flight data. We present extensive computational results comparing our methods and design choices in our heuristics. We compare also the results we obtain with the existing sectorization currently in use by the FAA.

5. We further extend our model for allowing $3D$ sectorization with *floors* and *ceilings*.

6. Finally we investigate methods for automating dynamic sector designs.

The GeoSect software is available online at [4]. Our static sectorization results are quite promising: our best heuristic methods yield an improvement by a factor between 2 and 3 over the current sectorization in terms of the time-average and the worst-case workloads of the maximum workload sector. An even better improvement is seen in the standard deviations (over all sectors) of both time-average and worst-case workloads. With the flow conforming methods, we show that while maintaining workload balance, we can still get good separations from crossing flows, and also get good angle of intersections of flows with sector boundaries. Extension to local methods for dynamic re-sectorization also leads to promising future directions.

The experiments currently include only en route airspace. Sectorization involving Terminal Radar Approach Control (TRACON) areas near major airports is envisioned as a further extension to the methods. We model workload in terms of aircraft density (number of aircrafts in a sector), determined by a given set of track data, which may come from historical data or from the results of a simulation. We have extended the results to include coordination workload in the objective function as well, so that we take into account the number of times a flight must be handed off between sectors.

The organization of material in this dissertation is as follows. Chapter 2 includes details of static sectorization methods, both for convex sector designs and flow conforming sector designs. In there, we also describe methods for $3D$ sectorization. Chapter 3 is dedicated to novel ideas for dynamic sector design, where the sector boundaries change over the course of the day. In Chapter 4, we explore two problems (re-scheduling aircraft and trajectory clustering) closely related to sectorization.

2 Static Sectorization

The word *static* suggests that the sector design remains fixed over the entire time period for which it was designed, typically a day or the entire season. The air traffic pattern for the whole time period is used to evaluate the workload and the goal is to balance this workload across different sectors. More formally,

Problem Statement The (static) *sectorization problem* is to determine a decomposition of a given airspace domain $\mathcal{D}$ into a set of k *sectors*, $\sigma_1, \ldots, \sigma_k$, in an "optimal" manner. Optimality is defined in terms of the *workloads*, $w(\sigma_i)$, of the sectors, where $w(\sigma_i)$ is a numerical value indicating the amount of "effort" required to manage and control traffic in sector σ_i. The objective may be to minimize the maximum workload (min-max) or to minimize the average workload (min-avg) across sectors, subject to an upper bound, k, on the number of sectors. Alternatively, the objective may be to minimize k subject to a bound on the maximum or average workload across sectors.

2.1 Convex Sector Design

In this section, we discuss sectorization methods where the resulting sectors are required to be convex. We assume that the given airspace domain is also convex to begin with. Though convexity is a good property for sector shape, in subsequent sections we relax this for more important features essential for "acceptable" sector designs. This is joint work [11] with Amitabh Basu.

We begin with a study of the 1D problem, which has interesting algorithmic aspects of its own. Further, the 1D solution is used within the 2D heuristics we develop and implement.

2.1.1 The 1D Sectorization Problem

Consider an airspace domain that is 1-dimensional, consisting of an interval, without loss of generality $\mathcal{D} = [0, 1]$, on the x-axis. Flights can take off at some point ("airport") of $\mathcal{D}$ and land at another point ("airport").

The input data consists of a set S of flight trajectories, each represented by a sequence of "waypoints", (x_i, t_i), where t_i is the timestamp when the flight is recorded to be at location $x_i \in [0, 1]$. We consider there to be a finite time horizon, $[0, T]$, containing all of the timestamps t_i. We generally assume that the flight speed between waypoints is constant; thus, a trajectory can be thought of as a t-monotone polygonal chain in the (x, t)-plane. If we view this problem in a "LineLand" model, then it makes sense that the trajectories be x-monotone as well; if the speeds are constant along each such trajectory, then the trajectories are simply line segments. (Other waypoints between the start and destination x-coordinates may be used to specify changes in speed or direction. If the 1D problem arises as a projection of the 2D problem onto the (x, t)-plane, the trajectories will, in general, zig-zag, not necessarily being monotone in x.) While our methods for the 1D problem can be extended to more general polygonal trajectories, here, we consider the case of the 1D problem in which the input $S = \{s_1, \ldots, s_n\}$ is a set of line segments in the (x, t)-plane, all of which lie within the 1-by-T rectangle, $[0, 1] \times [0, T]$.

The sectorization problem asks us to partition $[0, 1]$ into a set of k sectors, $\sigma_1, \sigma_2, \ldots, \sigma_k$; i.e., we desire partition points, $x_0 = 0 < x_1 < x_2 < \cdots < x_{k-1} < x_k = 1$, which define the sector intervals $\sigma_i = (x_{i-1}, x_i)$.

The *max-workload*, $w(\sigma_i)$, of a sector $\sigma_i = (x_{i-1}, x_i)$ is defined to be the maximum number of flights ever simultaneously in sector σ_i: this is given geometrically by the maximum number of segments of S intersected by a horizontal segment, $\overline{(x_{i-1}, t)(x_i, t)}$, for $t \in [0, T]$. One can envision a sweep of the rectangle $[x_{i-1}, x_i] \times [0, T]$ by a horizontal segment – the max-workload of σ_i is the maximum number of segments of S intersected during the sweep. The *avg-workload*, $\bar{w}(\sigma_i)$, of a sector $\sigma_i = (x_{i-1}, x_i)$ is defined to be the *time-average* number of flights in the sector σ_i: this is given geometrically by the sum of the lengths of the t-projections of segments S clipped to the rectangle $[x_{i-1}, x_i] \times [0, T]$, divided by T. If we let $\xi_i(t)$ denote the number of segments of S crossed by the horizontal segment $\overline{(x_{i-1}, t)(x_i, t)}$, then $w(\sigma_i) = \max_{t \in [0,T]} \xi_i(t)$ and $\bar{w}(\sigma_i) = \frac{1}{T} \int_0^T \xi_i(t) dt$.

The *min-k sectorization problem* is to determine a set of partition points x_i (and corresponding sectors σ_i) in order to minimize the number, k, of sectors in a partitioning of $[0, 1]$, subject to a specified *workload bound*, B. The workload bound B stipulates that $w(\sigma_i) \leq B$, or that $\bar{w}(\sigma_i) \leq B$, for all $i = 1, \ldots, k$, in

the max-workload or the avg-workload case, respectively.

The *min-B sectorization problem* is to determine a set of partition points x_i (and corresponding sectors σ_i) in order to minimize the upper bound, B, on the workloads of the sectors, subject to their being at most (and therefore exactly) k sectors, where k is specified as part of the input. In other words, we want to determine the x_i's, $i = 1, \ldots, k$, subject to $w(\sigma_i) \leq B$, or $\bar{w}(\sigma_i) \leq B$, for all $i = 1, \ldots, k$, in the max-workload or the avg-workload case, respectively.

Thus, we get four versions of our sectorization problem, depending if we are using max-workload or avg-workload measures, and depending on the choice of min-k or min-B in the optimization.

min-k, max-workload. We are given a budget B on the max-workload in each sector and wish to minimize the number, k, of sectors. We prove that the following greedy algorithm is optimal: At stage i, with partition points $x_1, \ldots, x_i$ already determined, we compute partition point x_{i+1} in order to make sector $\sigma_{i+1} = (x_i, x_{i+1})$ as large as possible, subject to the budget constraint B.

The determination of x_{i+1} according to this greedy rule is an interesting geometric subproblem in its own right, and it is related to the following problem: *Given a set of n line segments in the plane, determine the lowest point of the B-level.* Recall that the j-level of a set of line segments S is defined to be the locus of all points on S that have exactly j segments lying strictly below. In our setting, "below" means "leftward" in the (x, t)-plane, and "lowest" point on the B-level means the leftmost point of the B-level. The lowest point on the B-level in an arrangement of lines is solved in expected time $O(n \log n)$ by the randomized algorithm of Chan [19]. In fact, this algorithm is readily adapted to give the same expected running time $O(n \log n)$ for computing the lowest point on the B-level in an arrangement of line segments or x-monotone curves of constant complexity [20]. Below, we give a simple $O(n \log^2 n)$ deterministic algorithm; we are not aware of an $O(n \log n)$ deterministic algorithm for computing the lowest point on the B-level of an arrangement of lines or of segments.

Consider sweeping a vertical line ℓ rightwards from $x = x_i$. The max-workload of the sector between $x = x_i$ and ℓ can change only at certain events,

when ℓ passes over a *critical point*, and it can only go up (by definition) (see Figure 2). Each left endpoint of a segment of S is a potential critical point. A critical point may also occur at the intersection of two segments of S, if the signs of these segments' slopes are opposite (since, in this case, the t-projections of the segments within the vertical strip start to overlap, possibly causing the max-workload to change). A critical point may occur at the intersection $\rho_j \cap s_l$, for some segment $s_l \in S$, if the signs of the slopes of s_j and s_l are opposite; here, ρ_j is the rightwards ray from the right endpoint of segment $s_j \in S$. Finally, a critical point can occur at the intersection $\rho_{ij} \cap s_l$, for some segment $s_l \in S$, if the signs of the slopes of s_j and s_l are the same. Here, ρ_{ij} is the rightwards ray from the point $a_{i,j} = \{x = x_i\} \cap s_j$ on s_j intersected by the vertical line $x = x_i$.

We can now solve the geometric subproblem using binary search on the set of x-coordinates of potential critical points. Using slope selection (see Cole et al. [21]), we can, in $O(n \log n)$ time, compute the median x-coordinate, x', among vertices in the arrangement, $\mathcal{A}$, of the n lines containing each segment of S, the (at most n) lines containing each ray ρ_j, the (at most n) lines containing each ray ρ_{ij}, and the (at most n) vertical lines through left endpoints of segments in S. In fact, we compute x' to be the median x-coordinate among vertices of the arrangement that lie between $x = x_i$ and $x = 1$. Now, we can "test" the value x', to see if x_{i+1} should lie to its left or its right, by computing the workload, $w([x_i, x'])$: If $w([x_i, x']) > B$, then we know that $x_{i+1} < x'$; otherwise, $x_{i+1} \geq x'$. Computing the workload $w([x_i, x'])$ is easily done in time $O(n \log n)$, e.g., by clipping the segments S to the strip $[x_i, x']$, projecting the clipped segments onto the t-axis, and sweeping in t to determine the depth of overlap among the projections. Since there are at most $O(n^2)$ candidate critical points, and each step of the binary search takes time $O(n \log n)$, we get that the overall algorithm to determine x_{i+1} greedily takes (deterministic) time $O(n \log n \log n^2) = O(n \log^2 n)$. Doing this for each stage of the greedy algorithm yields the following:

Theorem 2.1. *The one-dimensional min-k, max-workload, sectorization problem can be solved exactly in (deterministic) time $O(kn \log^2 n)$, where k is the output optimal number of sectors. Using a randomized algorithm, it can be solved in expected time $O(kn \log n)$.*

Proof. We have described the algorithm and its running time already. In order to justify the correctness of the algorithm, consider an optimal partition $X^* =$

$\{x_1^*, x_2^*, \ldots, x_{k^*}^*\}$. Let the output of the greedy solution be $X = \{x_1, x_2, \ldots, x_k\}$. Let i be the first index for which $x_i^* \neq x_i$. If $x_i^* > x_i$, then x_i could not have been the greedy output, since we could have pushed x_i further to the right (to x_i^*) without violating the budget constraint B. Thus, $x_i^* < x_i$. Now, we can replace x_i^* with x_i in X^*. The workload of the sector $[x_{i-1}^* = x_{i-1}, x_i^* = x_i]$ clearly cannot exceed the budget B (since the greedy sectors must be feasible), and the workload of the sector $[x_i^*, x_{i+1}^*]$ only went down with the replacement of x_i^* with $x_i > x_i^*$. Continuing this argument, we convert solution X^* into solution X, proving that the greedy algorithm produced an optimal partition. □

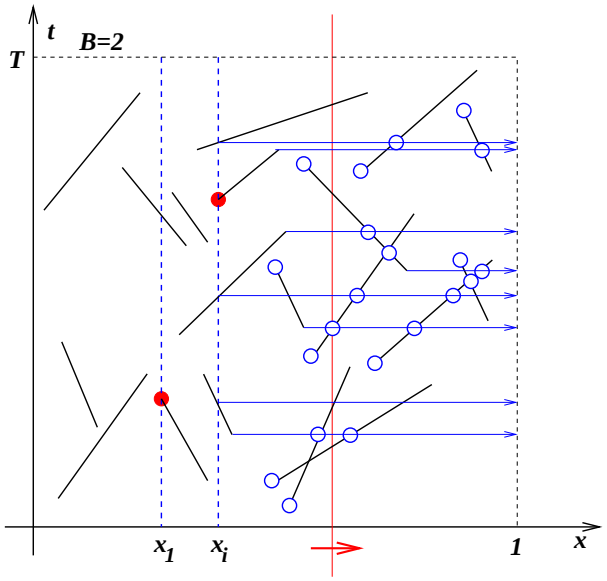


Figure 2: Sweeping ℓ (red) rightwards. The hollow (blue) circles indicate critical points where the max-workload might increase as ℓ sweeps.

min-B, max-workload. We are given an allowed number k of sectors and wish to determine a set of partition points, $x_1, \ldots, x_{k-1}, x_k = 1$, of $[0, 1]$ in order to minimize the maximum workload, $B = \max_i w(\sigma_i)$. We do this optimization using binary search, using the min-k solution above to test a particular value, B', of (integer) budget B. Note that the optimal B^* must lie between 1 and $B_0 \leq n$, where B_0 is the maximum number of segments of S intersected by

a horizontal line. For each test value B', we run the greedy algorithm to determine the optimal number of sectors, $k^*(B')$, subject to budget B'. If $k^*(B') > k$, then we know that $B^* < B'$; otherwise, we know that $B^* \geq B'$. The binary search concludes in $O(\log n)$ steps, so we get

Theorem 2.2. *The one-dimensional min-B, max-workload, sectorization problem can be solved exactly in (deterministic) time $O(kn \log^3 n)$. Using a randomized algorithm, it can be solved in expected time $O(kn \log^2 n)$.*

min-k and min-B, avg-workload. In the average workload case, we consider the "cost" of a sector to be the time-average number of aircraft in a sector. Since the time-average $\bar{w}(\sigma_i)$ for sector $\sigma_i = (x_{i-1}, x_i)$ is simply the sum, $(1/T)\sum_{s \in S} \mu_{\sigma_i}(s)$, of the lengths $\mu_{\sigma_i}(s)$ of the t-projections of the segments $s \in S$ clipped to sector σ_i, each of which varies linearly with x_i, we see that the function $f(x) = \bar{w}((x_i, x))$ that measures the time-average workload of the interval (x_i, x) is a piecewise-linear (and continuous) function of x. The function $f(x)$ has breakpoints that correspond to the x-coordinates of endpoints of S. For the min-k avg-workload, k is exactly equal to $\lceil (1/T)\frac{\sum_{s \in S} \mu(s)}{B} \rceil$, where $\mu(s)$ is the length of the t-projection of segment s. The sector (interval) boundaries can be determined by greedily scanning from left to right the $O(n)$ possible critical values of x, between which the function $f(x)$ has an easy-to-describe (linear) formula, which we can threshold against the budget B. Thus, the overall running time becomes just $O(n \log n + k)$ for the min-k problem. For the min-B version, the avg-workload of each of the k sectors will be exactly $(1/T)\frac{\sum_{s \in S} \mu(s)}{k}$, and the running time of the algorithm to determine the sector boundaries remains the same, i.e., $O(n \log n + k)$. The correctness of the greedy approach is proven similarly as before and is omitted here. In summary,

Theorem 2.3. *The one-dimensional min-k (and min-B), avg-workload, sectorization problem can be solved exactly in time $O(n \log n + k)$, where k is the output optimal number of sectors.*

Remark. Note that the min-B problem is (trivially) always feasible, both for max-workload and for avg-workload. The min-k problem is always feasible for avg-workload and, for max-workload, it is feasible and results in a finite k provided that B is at least as large as δ_{max}, the maximum number of segments of S passing through a common point. (If $B < \delta_{max}$, no partitioning in the immediate x-vicinity of the high-degree vertex will suffice to meet the (max-workload) budget constraint; if $B = \delta_{max}$, then there needs to be an infinite

sequence of partition points, converging on the x-coordinate of the high-degree vertex.)

2.1.2 The Sectorization Problem in 2D

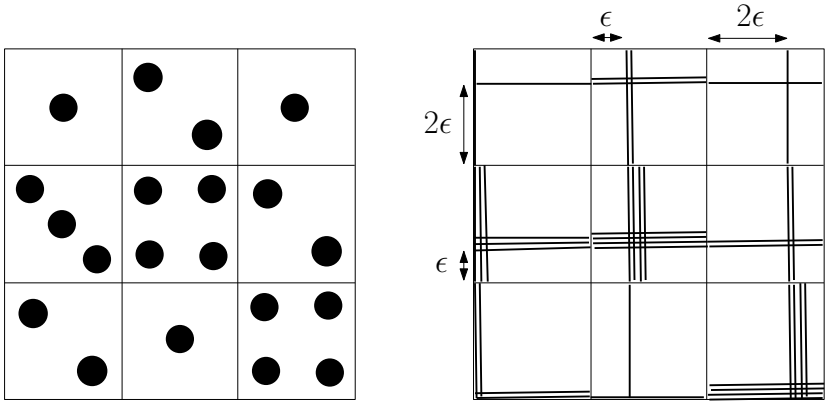
In contrast with prior work on partitioning sets of (static) points in the plane, or elements of an array (e.g., see [34, 35, 44, 45]), our sectorization problem involves a third dimension (time): The input data consists of a set S of trajectories, which correspond to t-monotone polygonal chains in (x, y, t)-space. We let n denote the number of trajectories, and N the total number of waypoints (vertices) in the full set of n trajectories. Given a domain of interest, $\mathcal{D} \subset \Re^2$, we are to partition it into a small number of sectors, each of which has a small workload. As in the 1D problem, we can distinguish the min-k from the min-B problem, where k denotes the number of sectors in the partition and B denotes an upper bound on either the max-workload or the avg-workload of the sectors.

The max-workload for a sector $\sigma \subset \mathcal{D}$ is the maximum number of trajectories intersected by a "horizontal" (in t) polygon of shape σ, sweeping vertically through time, $t \in [0, T]$. Another way to view the problem is to clip the 3D trajectories to the vertical cylinder defined by σ, and project each clipped trajectory onto the t-axis. The maximum depth of this set of intervals is the max-workload for σ; the sum of the interval lengths, divided by T, is the avg-workload for σ.

Hardness We expect that the sectorization problem in two (or more) dimensions is NP-hard for most formulations of the problem. Here, we prove hardness of the special case in which sectors are required to be axis-aligned rectangles, and the goal is to minimize the max-workload upper bound B, subject to a bound k on the number of sectors. Hardness follows from the result of Khanna, Muthukrishnan et al [34], who proved that the following problem is NP-hard (and also NP-hard to approximate within a factor of $\frac{5}{4}$): Given an $n \times n$ array A of integers, find a rectangular partition of A into k rectangles, in order to minimize the maximum weight of a rectangle. The weight here is defined to be the sum of the array elements in the rectangle.

For a given instance of the array partitioning problem, we construct an instance of the sectorization problem in the following manner. Consider a two-dimensional $n \times n$ grid in the (x, y)-plane corresponding to the array A, with unit width for each cell. Let $\epsilon = \frac{1}{n}$. For each cell (i, j) of the array A

$(1 \leq i \leq n, 1 \leq j \leq n)$ we denote the weight of the cell by $w_{i,j}$. In the cell corresponding to (i, j), we put $w_{i,j}$ tracks going from the left boundary to the right boundary in the time interval $[0, 1]$ and $w_{i,j}$ tracks going from the bottom boundary to the top boundary in the time interval $[1, 2]$. The horizontal tracks are at a distance of $(i-1)\epsilon$ from the bottom boundary; similarly, the vertical tracks are at a distance $(j-1)\epsilon$ from the left boundary of the cell. See Figure 3.



Original Array A Dark lines show tracks

Figure 3: Reduction from array partitioning to rectangular partitioning for a 3×3 array.

For any solution for the array-partitioning problem, it is easy to see that there exists a corresponding solution of the rectangular-partitioning problem that gives the same solution in terms of workload. We need to show that any solution of the rectangular-partitioning problem yields a solution for the array-partitioning problem. First, we observe that if one of the rectangles in the solution has vertical tracks from two horizontally adjacent cells corresponding to $(i, j), (i, j+1)$, then its workload is at least $w_{i,j} + w_{i,j+1}$. This is because the distance between the "bundles" of vertical tracks are at a distance of $1 + \epsilon$, so this implies that there is a time instant $t \in [0, 1]$ such that the horizontal tracks from both cells are present in this rectangle. Similarly, if horizontal tracks are present in a rectangle from two vertically-adjacent cells corresponding to $(i, j), (i+1, j)$, then the rectangle's workload is at least $w_{i,j} + w_{i+1,j}$. It is clear that if horizontal tracks are present from two horizontally-adjacent cells, then the workload is either the sum or is equal to that of one of the cells; a similar

statement applies to vertical tracks from two vertically-adjacent cells. The above discussion implies that we can always convert a rectangular-partitioning solution to one that conform to the boundaries of the grid, which then gives a solution to the array-partitioning problem. Thus, we have the following theorem:

Theorem 2.4. *The optimal sectorization problem (min-k or min-B) for partitioning into rectangular sectors in two dimensions is NP-hard.*

Recently, Farrahi et.al. [27] show that ASP is NP-hard in general.

Heuristics for 2D Sectorization Given the difficulty of solving the 2D sectorization problem exactly, we turn our attention to heuristics for its solution. We consider the min-k version, in which a budget B is given, and our goal is to partition $\mathcal{D}$ into a small number k of sectors.

Our heuristics for 2D sectorization are based on two forms of recursive partitioning: binary space partitions (BSP) and *pie-partitions*. BSP algorithms have been studied extensively in the computational geometry literature, starting with the work of Paterson and Yao [46, 47]. Pie-partitions are based on a multi-way partition into cones having a common apex; see below. All of our heuristics have the property that they guarantee convex sectors when applied to a convex domain $\mathcal{D}$.

The use of recursive partitions heuristics is both natural and theoretically motivated. For sectorizations based on BSP partitions whose cuts come from fixed orientations (as ours do) with discretized intercepts (translations), we are able to solve the min-k problem (for given budget B) optimally, as well as the min-B problem (for given k) using dynamic programming. A subproblem is defined by a convex polygon having $O(1)$ sides; by selecting an optimal cut from among a discrete set of possibilities, and recursively optimizing on each side of the cut, we obtain an optimal BSP-based sectorization. This sketches the proof of the following theorem:

Theorem 2.5. *The min-k and min-B optimal fixed-orientation, discrete intercepts BSP sectorization problem in 2D has an exact polynomial-time algorithm.*

Proof. Let c be the number of fixed orientations and d be the number of fixed intercepts (which may depend on the endpoints of the tracks). This gives us $O(d^{2c})$ possible (convex) polygons using these orientations and intercepts,

since a polygon has at most two edges of any one of the c orientations. A subproblem of the dynamic program is such a polygon P, and we maintain the optimal way to partition P in an array. The algorithm for min-k, with given budget B is as follows (it returns the number of sectors):

Partition_mink(Polygon P)

(i) If the max-workload of the polygon is B, simply return 1.

(ii) Else, for each pair (o, i) of orientation and intercept, recursively solve the subproblems corresponding to the two polygons (say P_1 and P_2) into which P is divided by the cut corresponding to (o, i), and compute $w(o, i)$ = Partition(P_1) + Partition(P_2).

(iii) Return $w(o, i)$, which is minimum over all choices of orientation and intercept of a partitioning cut.

For min-B, given k we similarly have the following algorithm to compute optimal workload:

Partition_minB(Polygon P, k)

(i) If $k = 1$, simply return max-workload(P).

(ii) Else, for each pair (o, i) of orientation and intercept, and every possible way to partition k into k_1 and k_2 such that $k = k_1 + k_2$, recursively solve the two polygons (say P_1 and P_2) into which P is divided by the cut corresponding to (o, i), and compute $B(o, i, k_1, k_2)$ = max{Partition(P_1,k_1), Partition(P_2,k_2)}.

(iii) Return $B(o, i, k_1, k_2)$, which is minimum over all choices of orientation and intercept of a partitioning cut, and k_1 and k_2.

It is easy to see why the above algorithms work. The first cut made by an optimal solution on the polygon P, is one of the cuts that is considered by the algorithm, and then the subproblems are recursively solved. Now look at the optimal solution on either side of this cut. The subproblems solved recursively on either side can be only better than this optimal solution. Moreover, the first cut found by the algorithm did at least as well as this (optimal) cut. So the output from the algorithm is at least as good as the optimal partitioning.

The running time of each algorithm is clearly polynomial in N (the total complexity of the input trajectories) and d, for fixed c: There are $O(dc)$ candidate cuts for each of $O(d^{2c})$ subproblems, and the evaluation of the workload associated with a subproblem can be computed readily by truncating each trajectory at the boundary of the subproblem and projecting onto the time axis. □

If we do not restrict ourselves to BSP sectorizations, but still consider the class of allowable cuts to lie on a discrete set of lines, of fixed (c) orientations and discrete intercepts, then we can obtain a polynomial-time approximation algorithm for the (non-BSP-based) min-k sectorization problem, using the fact that an optimal sectorization can be converted into a BSP sectorization with a small factor increase in the number of sectors: We simply apply the dynamic programming algorithm, as above, to find the best BSP-based sectorization, and then appeal to the known results on the size of a BSP partition of a set of (convex) objects to argue that the best BSP-based sectorization yields a number of sectors that is within a factor of the number of sectors in an optimal (not necessarily BSP-based) sectorization. In particular, this yields a 2-approximation for the rectangular (axis-parallel) case, by the results of [14] on the BSP of a packing of axis-aligned rectangles.

Throughout our discussion below, we will interchangeably use the term "weight" and "workload" when referring to a sector.

BSP Heuristic Rather than implementing a relatively high-degree ($d^{O(c)}$) polynomial time dynamic programming algorithm, as described in the previous section, we have chosen to craft BSP heuristics based on computing a *most balanced cut* at each stage, which is defined as follows. Given a node of the current BSP subdivision, with associated sector σ, our algorithm finds a straight cut (from among a set of fixed orientations) to partition the convex polygon σ into two subpolygons, in order to minimize the maximum workload (either max-workload or avg-workload) of the two subpolygons. This strategy leads to the following simple consequence in the avg-workload case about the relative weights (workloads) of the sectors: In the final sectorization using most balanced cut BSP, the ratio of the (avg-workload) weight of the heaviest sector to the lightest sector is at most 2. In our experiments, as we describe later, we have further refined the most-balanced cut method for avg-workload in order to partition avg-workload exactly across the k sectors, while simultaneously attempting to control the max-workload balance; see Section 2.1.3.

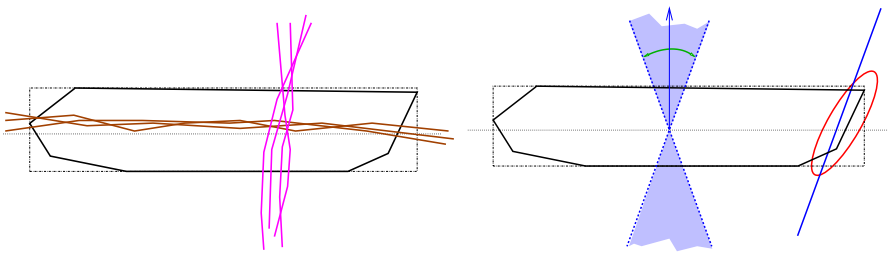
In order to find the most balanced cut, we use a discrete set of c *allowable orientations* for our cut. For each orientation, we find the most balanced cut with that orientation as follows. We project the line segments that make up the trajectories onto a plane perpendicular to the cut orientation, resulting in the 1D problem. We now use a binary search on the critical points (as defined in the previous section) to find the most balanced cut in the 1D case. Thus, each step of the BSP takes worst-case time $O(N^2c)$: $O(N)$ for projecting the segments, $O(N^2)$ for finding the critical points (which can be found in output-sensitive time, by standard techniques), and then finally the binary search for the most balanced cut. If we finally end up with K sectors, the entire procedure takes worst-case $O(KN^2c)$ time (again, with corresponding speed-up for using an output-sensitive segment intersection algorithm).

Since the calculation of the critical points becomes the bottleneck in this heuristic, even if using clever means of implementing nearly output-sensitive algorithms, in our experiments we decided to use a coarser set of points to search for the cut. We refer to this set as the *approximate critical set*. We empirically decide the coarseness of this set and prove experimentally that for the real track data, this works just as well (in practice) as the original set of critical points and saves tremendously on the execution time.

Avoiding Bad Aspect Ratio of Sectors The balanced BSP heuristic can clearly produce very skinny sectors, even while producing sectors with well-balanced workloads. Skinny sectors can be undesirable because air traffic passing through the sector perpendicular to the its diameter will have very small dwell time (time for which the flight remains within the sector). Figure 4 (left) shows the behavior of traffic with skinny sectors.

To get a sector whose shape is invariant to the direction of air traffic, we use the *aspect ratio* of the region we are subdividing to guide us. For any rectangle, define α to be the ratio of the smaller side to the larger side. (Note that aspect ratio is often defined to be $1/\alpha$.) For any sector σ that we want to subdivide by the most balanced cut, we also use the following constraint for the cut. Consider a bounding rectangle with the smallest α for the region. We only consider cuts with an orientation within a small range of the orientation of the smaller side of this rectangle. In our implementation, we use a range of $[\theta - \alpha\frac{\pi}{2}, \theta + \alpha\frac{\pi}{2}]$, where θ is the orientation of the smaller side. However,

this heuristic can still result in bad aspect ratio for a sector. Refer to Figure 4 (right).



(a) The (magenta) air traffic remains in the sector for very short duration (bad), while the (brown) air traffic remains for long duration (good).

(b) The allowable range for cut orientations is shaded. Circled (red): a cut within the allowable orientation range may result in a skinny polygon on the right.

Figure 4: Modeling aspect ratio constraint.

We suggest another heuristic to circumvent this problem. We define $\beta < 0.5$ to be a user-specified lower bound on $\alpha(\sigma)$, which our algorithm is expected to respect in its decomposition. Given an orientation for the cut, we have to find a range for the cut so that the resulting two polygons have $\alpha \geq \beta$. In our experiments we naively search for this range by linearly searching through the approximate critical set. This range may clearly not exist (for example, set $\beta = 0.8$ and consider a square – no range exists for any orientation). However, for reasonable values of β this seems to work quite well. Empirically, we observed that for $\beta < 0.5$, if the original polygon has $\alpha \geq \beta$, this heuristic works extremely well.

Pie Cutting In addition to the BSP cuts, we consider another cutting operation to allow for more flexibility during sectorization. This is the so-called "pie-cut". For this we fix a point within the region (called the *center*) and an orientation. We now wish to make a pie-cut which comprises three rays originating from the *center*. One of the rays is along the designated orientation. The other two are such that the resulting 3 pieces are all convex and as well-balanced as possible. We accomplish this pie-cut in two steps, obtaining one cut in each step. The line segments are first transformed to their polar coordinates, in the following sense. Consider any point p with polar coordinates

(r, θ, z) with respect to the *center* and the given orientation (r is the distance from the *center*, θ is the angle that the line through p and the *center* makes with the given orientation. This point is transformed to (θ, z), resulting again in an instance of the 1D sectorization problem. Then a cut is found that divides the workload in the ratio $1 : 2$. Then the second cut is chosen with range restrictions (so that the resulting regions are convex) to balance the workload in the $\frac{2}{3}$-sized region. See Figure 5.

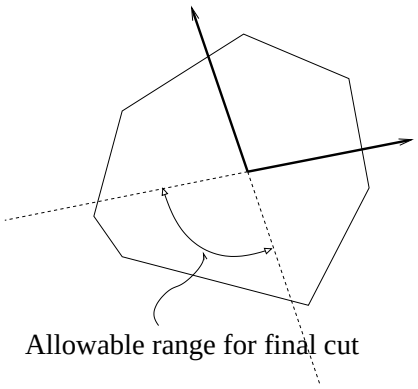


Figure 5: Range constraints for pie cutting.

For greater flexibility and control over α, we also use the pie-cut operation with more than 3 cuts. Note that it is not desirable for many sectors to meet at one point (pie-cut center here) as it decreases the chance of an airplane to stay in a sector for reasonable time, before leaving it. So there has to be an upper-bound on how many cuts should be allowed in pie-cut operation and it can easily be added as a constraint. If the current workload is W, and $P = \lfloor W/B \rfloor$, we start with $max(P, 5)$ cuts and if any of the resulting regions has α less than the threshold, we try with one less cut and so on, until we reach 3. At 3 however, even if one of the α values are bad, we make the cut anyway to maintain convexity of the regions. This is the disadvantage of pure pie-cuts. This can be remedied to a large extent if we combine BSP cuts with Pie-Cuts. That is our motivation for the final heuristic.

Wheel Cuts As noted above, it is not desirable for many sectors to meet at one point (eg. pie-cut center), we introduce another kind of partitions: Wheel-Cuts. In order to partition a region into $k + 1$ subregions we first start

with a pie-cut partition into k subregions, and then replace the central node with a convex k-gon whose vertices lie along the rays of the pie-cut partition. Wheel-cuts result in a partition all of whose nodes have degree 3, typically with no small angles. Further, the degree-3 nodes are well separated. Three examples are shown in Figure 6.

The central k-gon of a wheel cut can be chosen in order to balance exactly the workload in all $k+1$ subregions; however, for simplicity in our implementation, we have opted to construct wheel-cuts more directly from pie-cuts: we begin with a pie-cut into k subregions that exactly balances workload, then we replace the central node with a convex k-gon whose vertices are each chosen to be at the same point, proportionally, along the ray segments of the pie-cut, with the size of the central region chosen to give it exactly $1/(k+1)$ of the average workload. This simplification allows us to adjust a single parameter for the load balance of the central region (versus k parameters if we adjust each vertex of the central region independently along its pie-cut ray). However, the resulting workload values of the k subregions surrounding the central region will not, in general, be perfectly load balanced with respect to the workload in the resulting wheel-cut.

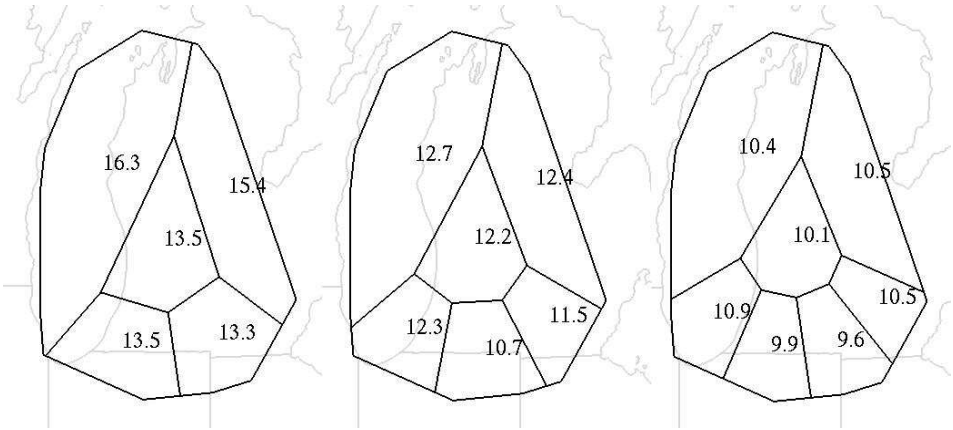


Figure 6: Example showing Wheel-cuts of the region into 4,5 and 6 pieces. The numbers show time-avg. workload.

The Final Heuristic We formulate a method using the operations of BSPs and Pie-cuts (Wheel-cuts may also be used here). The final heuristic first attempts to make a possible pie-cut. If the pie-cut is unable to find a partition respecting the β threshold, we use a BSP cut. The new regions are then inserted into a priority-queue according to the workloads. We recurse on the heaviest region until all the regions have a workload less than B.

Other heuristics Some other heuristics for 2D-partitioning are conceivable. For example, a partitioning resembling the Voronoi regions of some predetermined centers. There does not seem to be sufficient evidence to suggest that such combinatorial structures will optimize the workload as considered. A partitioning that does load balancing amongst different sectors according to the definition of workload in this paper does not seem to have any similarity to the structure of Voronoi regions. We feel our heuristics are natural strategies to try and implement when trying to optimize a function like the workload. A related, but perhaps of not much relevance, clustering idea appears in [31].

2.1.3 Experimental Results

Experimental Setup Implementation of the above heuristics (GeoSect1.0) is done in C++ (Microsoft Visual Studio 6.0), using the OpenGL Graphics library for all visualizations. All three heuristics (BSP, Pie-Cuts and the Final Heuristic) were implemented as described above. We also implemented the capability to search over an *approximate critical set* of points while solving the 1D problem to save some time while not compromising much on the ability to balance workload. All experiments were run on machines with 3.2 GHz Pentium 4 processors and 1GB RAM.

Data is provided to us by Metron Aviation. The historical track data corresponds to a 25-hour period from 04:00, June 27 to 05:00, June 28, 2002, with 74588 flight tracks, and the average complexity (number of bends) of each track is 59.26. We compare our results to existing sector data. (We are not using ultra high-altitude sectors.) In evaluating sector workloads, both in our sectors and in the existing sectors, we are assuming that all track data is relevant to the sectors. Note that some fraction of the track data may correspond to ultra high-altitude sectors and may not be relevant to the workload of the high-altitude sectors. Another limitation of our test data is that it does not include a broad sample of different traffic patterns, which may be impacted, e.g., by weather events.

Tuning the Parameters of the Heuristic For best performance of our heuristics, we first tune the user-specified parameters that are used at each stage of the heuristic. For tuning parameters in our heuristic, we use 5 different sub-regions of the NAS, shown in Figure 7. The selection of the regions was based on a visual inspection of the track data to correspond to both high and low traffic regions.

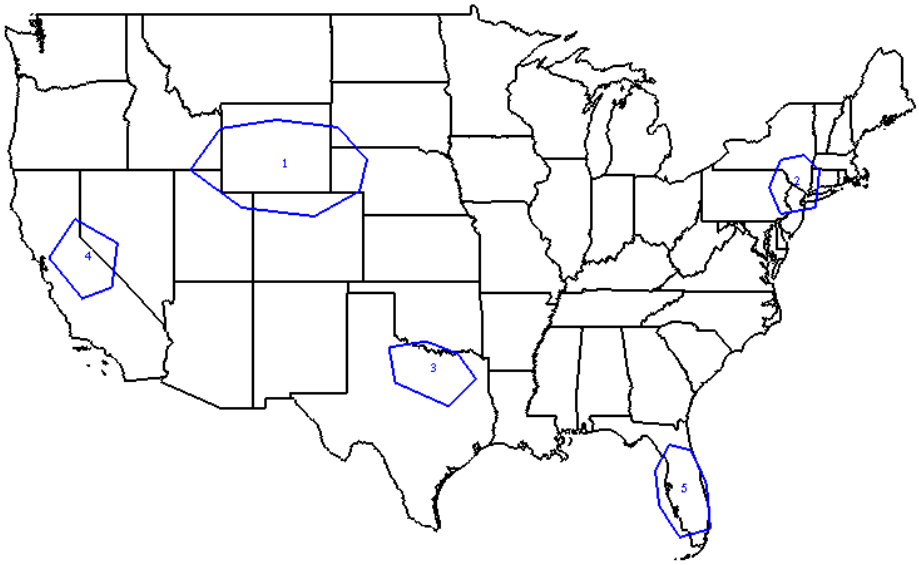
BSP We begin by selecting good choices of parameters for the BSP cuts. Statistics were generated for *Number of sectors* and *Max, Min, Average and Standard Deviation* for *Worst-case workload*, *Time-average workload*, and *aspect ratio* α, for each of the 5 sub-regions of the NAS (Figure 7, top). We summarize our experimental findings:

- *Number of discrete orientations while searching for the most balanced cut.* We generated the above statistics for the following set of values: {2, 4, 6, 8, 10, 12, 14, 16, 24, 32}. The statistics show that increasing the number of orientations beyond 10 does not yield any significant change in the results. We choose to use 16 orientations in all future experiments.

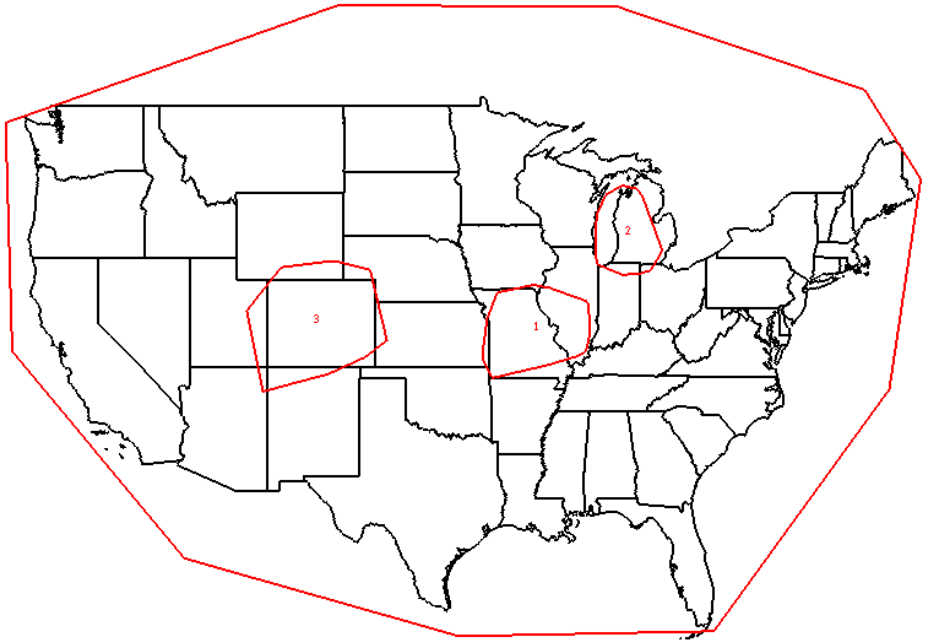
- *Discretization for balanced search in a given orientation.* Ideally, we should use the critical points of the projected tracks. However, as mentioned earlier, critical point computation is expensive and we use the *approximate critical set* instead. We examined the data for 5 different values of the discretization parameter (spacing between consecutive candidate cut lines): 0.1, 0.01, 0.001, 0.0001, 0.00001. For values less than 0.001, there is only a very slight fluctuation in the results. We pick value 0.0001 for our experiments.

- *Choice of β for aspect ratio control.* The goal is to obtain reasonably fat sectors without compromising too much on the quality of the sectorization (number of sectors, workload balance etc.). We experimented with 10 uniformly spaced values between 0.01 and 0.4 for the value β. Arbitrary small fluctuations are observed in the range 0.01 to 0.2. The experiments suggest predictable behavior for $\beta \geq 0.2$. In the results below (Table 3) we show the effect of different choices of β on the final workload balancing.

Some results are presented in Table 1. Here, we subdivide to balance the *worstcase workload*. We can see that the BSP cuts achieve highly balanced sectors in terms of workloads as reflected by the *standard deviation* of the



(a) For tuning the parameters.



(b) For testing the Final Heuristic.

Figure 7: Regions used for the experiments.

Region	$WorstcaseWorkload$			$TimeAverageWorkload$			α		
	Max	*Avg.*	*Std.Dev.*	Max	*Avg.*	*Std.Dev.*	*Avg.*	Min	*Std.Dev.*
1	5	4.237	0.501	0.891	0.344	0.133	0.572	0.350	0.139
2	5	4.571	0.564	0.893	0.403	0.151	0.609	0.350	0.156
3	5	4.559	0.537	0.766	0.409	0.138	0.593	0.350	0.148
4	5	4.442	0.573	1.206	0.405	0.169	0.591	0.350	0.141
5	5	4.414	0.553	0.771	0.351	0.142	0.587	0.350	0.149

Table 1: BSP results for the 5 regions after parameter tuning.

worstcase workload. We could instead choose to balance the time-average workload, which would result in better standard deviation statistics for the time-average case. By the nature of the heuristic, we have a guarantee on the minimum value of α.

Pure Pie-Cut The only decision parameter for this class of heuristics is how to choose the orientation for the first cut of the pie. We compared two choices: (1) Use the α restriction, as in BSP cuts, i.e. the first cut is approximately perpendicular to the diameter; and, (2) Choose a random orientation uniformly in $[0, 2\pi]$. The aspect ratio readings fluctuate unpredictably for both cases. This happens because 3-pie cuts can result in regions with very bad aspect ratio, as mentioned previously.

The Final Heuristic We use the parameter choices described in the previous two subsections (for BSP and Pie-Cut methods) in our experiments with the Final Heuristic.

Achieving the Workload Balance The main goal is to balance the time-average workload across all of sectors while controlling the worstcase workload and also the aspect ratios of the sectors. It is easy to see that one can exactly balance the time-average workload, i.e. given k, the number of sectors, it is possible to achieve sectors with workload exactly equal to the total workload divided by k. We control the worstcase workload by iteratively choosing the sector with the maximum worstcase workload as the candidate for splitting. Also, since we know the target workload for individual sectors, we restrict ourselves to cuts that preserve integer multiples of target workload on both sides. For example, if we want to split a region with time-average workload 9 into 3 sectors, we do not split it into 4.5-4.5; we instead restrict to cuts that split it in 3-6 or 6-3, so that later the sector with time-average workload

6 can be split in 3-3. The aspect ratio of sectors is controlled, as described previously, by avoiding the cuts that result in bad aspect ratios. Definitely, there is a trade-off between preserving good aspect ratio and optimal workload balance. This trade-off is indicated in the results in Table 3.

Comparing the Final Heuristic with Existing Sector Data To compare our Final Heuristic with the original sectors, we conduct experiments for two types of geographical domains: (1) ["Domain 1"] a specific convex polygonal region, C, selected to contain approximately 10 current sectors; and (2) ["Domain 2"] a large convex polygonal region, U, selected to contain all of the continental USA. To be as accurate as possible, we purposely select these regions so that they closely match existing boundaries of the sectors. See Figure 7.

When computing the statistics for original sectors, we consider only the sectors that are completely inside the domain of interest. While we compare the results for both time average workloads and worstcase workloads, the goal for each of the following experiments was to balance the time-average workload.

Results for "Domain 1" Both the BSP method and the Final Heuristic performed well in comparison with the original sector data. The statistics for one of the regions are shown in Table 2. Our heuristic achieves a significant improvement (by a factor of 10) over the original sectors in the standard deviation of the workloads and decreases substantially the maximum workload, while using the same number of sectors and having comparable average workloads. The average workloads are slightly higher for our heuristic because it is applied to a *convex* domain (Domain 1), which is slightly larger than the union of the original sectors. The average workload for original sectors is computed for only those original sectors which are fully contained within the domain. But, the average workload calculation for our sectors includes air traffic which may lie outside the original sectors (but within Domain 1). The BSP results are shown as well. This experimentally supports our claim that the heuristics achieve highly balanced workloads, while avoiding skinny sectors.

Results for "Domain 2" As with Domain 1, in the Domain 2 (the entire NAS) experiments, we computed statistics only for those original sectors that are fully contained within Domain 2. In running our methods, we used the same number (411) of sectors as there were original sectors in Domain 2 (ex-

Sectorization	Time Average Workload			Worstcase Workload			α		
	Max	*Avg.*	*Std.Dev.*	*Max*	*Avg.*	*Std.Dev.*	*Avg.*	*Min*	*Std.Dev.*
Original Sectors	12.33	6.899	2.578	44	26.8	8.340	0.548	0.264	0.169
Final Heuristic	7.289	7.226	0.054	30	27.9	1.221	0.534	0.323	0.159
BSP	7.9525	1.226	0.302	31	27	2.323	0.522	0.25	0.171

Table 2: The statistics for pure BSP, the Final Heuristic and the original sectors for Domain 1. The number of sectors was 10 in all cases.

cept that for Pie-Cut it was 412, due to the nature of the combinatorics of Pie-Cut partitions). Results are presented in Table 3.

Clearly the Final Heuristic and the BSP give very nicely balanced sectors, while avoiding skinny (low aspect ratio) sectors, since they are constrained to have aspect ratio $\alpha \geq \beta$. (For the original sectors, we list the value of β as 0, since there is no explicit aspect ratio bound on them.) Notice that as β is increased, our partitioning algorithms become more constrained, thereby decreasing their ability to achieve workload balancing (and increasing the standard deviations of the workloads).

The standard deviations of workloads produced by our methods are about an order of magnitude better than the standard deviation of the workloads for the original sectors. Also, the maximum value of worstcase and time-average workloads in the sectorizations produced by our methods is better than the corresponding values for the original sectors, by a factor between 2 and 3.

The Pie-Cut heuristic fails to keep the aspect ratio above the threshold and actually gives very poor values for α. Still, though, it does balance the workload better than the original sectors.

As with Domain 1, our average workloads are slightly higher for our heuristics than for the original sectors, since our sectorizations completely cover Domain 2, while the union of the original sectors for which workload is computed is a proper subset of Domain 2. (This under-counting should not have much impact on the variation in the workloads across sectors – the variation is the main subject of our investigation in load balancing.)

Our heuristics methods are thus seen to be very effective in global sectorization, in terms of balancing workload and producing sectors with good aspect ratio.

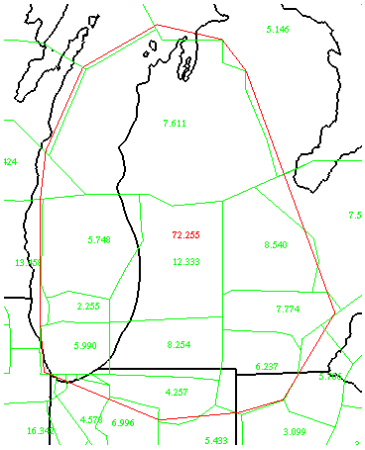
Sectorization	β	Time Average Workload		
	$(\alpha \geq \beta)$	Max	$Avg.$	$Std.Dev.$
Original Sectors	0	24.519	6.283	3.378
Final Heuristic	0.15	7.335	6.365	0.157
Final Heuristic	0.25	9.283	6.365	0.294
Final Heuristic	0.30	8.938	6.365	0.457
BSP	0.15	7.343	6.365	0.0715
BSP	0.25	9.568	6.365	0.426
BSP	0.30	9.545	6.365	0.512
Pie-Cut	0	11.085	6.35	2.901

Sectorization	β	Worstcase Workload		
	$(\alpha \geq \beta)$	Max	$Avg.$	$Std.Dev.$
Original Sectors	0	87	24.569	10.437
Final Heuristic	0.15	39	25.297	2.586
Final Heuristic	0.25	34	25.253	2.539
Final Heuristic	0.30	40	25.426	2.939
BSP	0.15	34	25.207	2.567
BSP	0.25	36	25.11	2.882
BSP	0.30	35	25.1	2.849
Pie-Cut	0	47	25.041	8.812

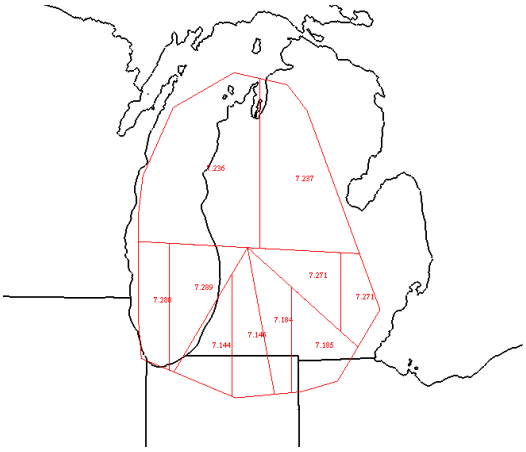
Sectorization	β	α		
	$(\alpha \geq \beta)$	$Avg.$	Min	$Std.Dev.$
Original Sectors	0	0.316	0	0.241
Final Heuristic	0.15	0.45	0.15	0.185
Final Heuristic	0.25	0.506	0.25	0.152
Final Heuristic	0.30	0.532	0.30	0.151
BSP	0.15	0.588	0.15	0.188
BSP	0.25	0.60	0.25	0.181
BSP	0.30	0.578	0.30	0.164
Pie-Cut	0	0.286	0.021	0.175

Table 3: The statistics for the original sectors, the Final Heuristic, pure BSP, and pure Pie-Cut for Domain 2. The number of sectors was 411 in all cases except Pie-Cut, for which it was 412.

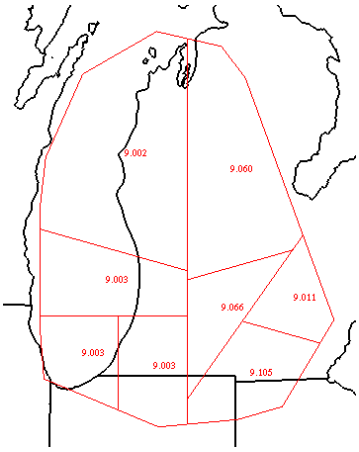
Refer to Figure 8 and Figure 9 for some screenshots of the sectorizations we compute.



(a) Original sectors.



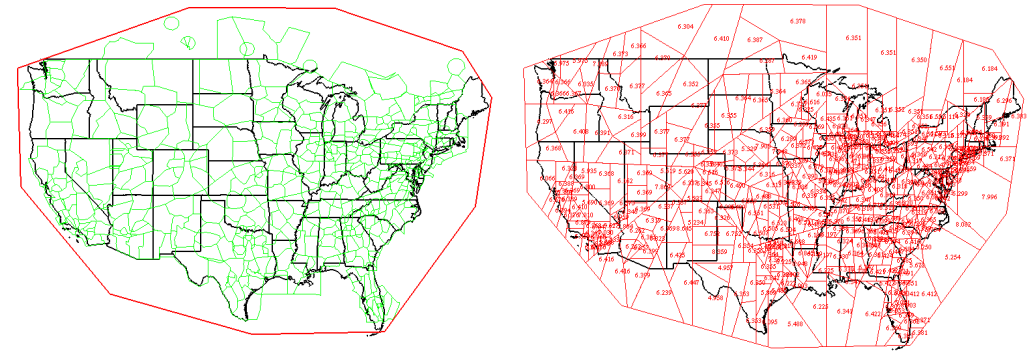
(b) Final Heuristic partition.



(c) Binary Space Partition.

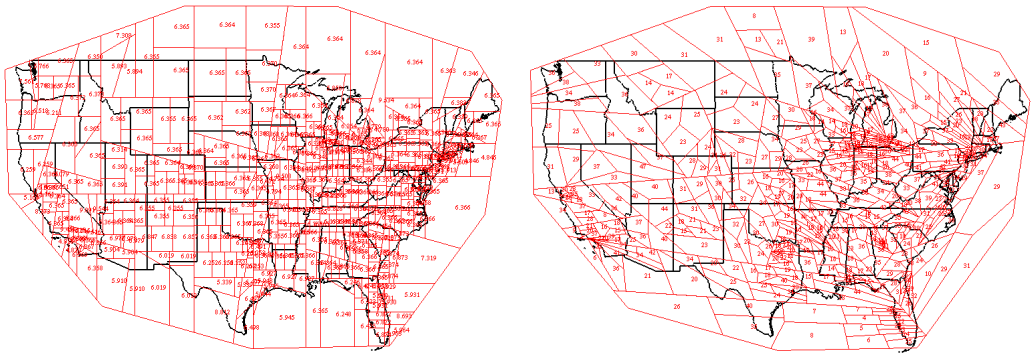
Figure 8: Partition results for Domain 1.

Comparison with a Mixed Integer Programming Method We have directly compared our Final Heuristic with a leading sectorization method based on formulating the problem as a Mixed Integer Program (MIP) [59]. The MIP method considers the domain of interest to be a union of small regu-



(a) Original sectors (411) strictly within the selected region.

(b) Final Heuristic partition (411 sectors).



(c) BSP (411 sectors).

(d) Pie-Cut partition (412 sectors).

Figure 9: Partition results for the continental USA (Domain 2).

Sectorization	No. of Sectors	Time Average Workload		
		Max	$Avg.$	$Std.Dev.$
IP Method	18	5.408	4.184	0.658
Final Heuristic	18	5.158	4.771	0.194

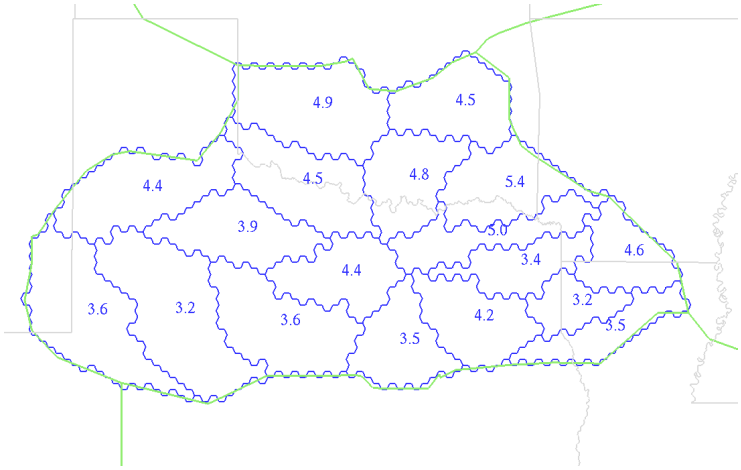
Sectorization	No. of Sectors	Worstcase Workload		
		Max	$Avg.$	$Std.Dev.$
IP Method	18	20	16.611	2.059
Final Heuristic	18	23	18.167	2.034

Sectorization	No. of Sectors	α		
		$Avg.$	Min	$Std.Dev.$
IP Method	18	0.442	0.210	0.148
Final Heuristic	18	0.600	0.319	0.173

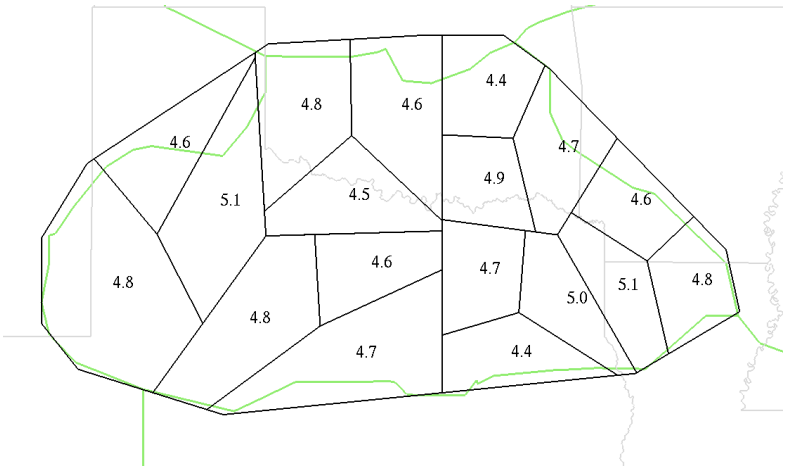
Table 4: The statistics for the MIP solutions and the Final Heuristic for sectorizing ZFW (Dallas) center.

lar hexagonal cells. It then formulates the optimization problem as a MIP for clustering cells in order to optimize the "coordination workload" (the number of times a flight crosses a sector boundary), while constraining the maximum deviation in the average workload per cluster (sector) of cells. In Table 4 we show the results of two methods for sectorizing ZFW (Dallas) center. The objective of the Final Heuristic in this experiment was to balance the average workload while keeping the aspect ratio of the sectors at least 0.30

We see that the Final Heuristic does a better job at balancing the average workload of the resulting sectors and keeping their aspect-ratios high. The MIP method, though, did a better job of minimizing the worstcase workload. One practical issue with the MIP method is that the resulting sectors have irregular boundaries, since the sectors correspond to unions of hex-cells; this is often addressed by doing a post-processing (polygonal simplification) of the sector boundaries, possibly at some cost in optimality. The running-time of the MIP method is also considerably higher than our methods described in this paper, since it relies on solving a complex MIP (which is done using CPLEX). Refer to Figure 10 for the screenshots of these comparison.



(a) MIP method.



(b) GeoSect: Final Heuristic partition.

Figure 10: Partition results for ZFW Center.

2.2 Feedback from the Controllers

To aid evaluating the quality of a sectorization, we showed the airspace designs obtained with above heuristics to controllers for feedback [42] on how well the objectives and constraints were being captured in the current model. We now discuss some of the features of GeoSect1.0 sectorizations that were flagged by controllers as problematic, and why they were not desirable. We have modified our model to account for these issues, and further experiments use the new model, with new algorithms for partitioning.

Before getting to details of the feedback, let us define a new term: *Dominant Flow* (or just flow). Each individual flight track may be considered as a flow, but the routes which encompass many flights bear more importance. These are called dominant flows. Often the air-plane routes are modified to conform to sector boundaries, which means if a dominant flow needs to be modified, many air-planes must be detoured. Thus, interaction of dominant flows with the sector boundaries is more critical. Section 4.2 is dedicated to identification of dominant flows from the track data. Henceforth, when we refer to flow, we mean dominant flow.

Straight-Line Cut Partitions (BSP) Controller feedback indicated that, for most of the resulting sectors in Figure 10, sector size was acceptable, but the corners were an issue. Referring to the lettered regions in Figure 11, controllers noted the following problems:

Region A: The critical question is the direction of the flow. If the direction of the flow is from NE to SW, then perhaps the sector should end perpendicular to that flow. The sector should not end in a tip, but have its tip blunted so that flow is orthogonal to the sector boundary. If the flow is E-W, then again, blunting off this tip will allow the flow to have a clear orthogonal sector line between the left and right sectors.

Region B: This is not a good intersection, since any aircraft flying near it or any flow near it could quickly pass between three sectors in minutes. It is not desirable to have four sectors meet like this. It is also suggested that degree 3 nodes are highly preferred to nodes of higher degrees and that one does not want two nodes of degree 3 to be too close to one another.

Region C: Traffic flow should again be considered in this region. The sector

in this region has a very acute angle in the upper NE corner, which is a problem if an aircraft quickly crosses over it, since there is not enough time to perform coordination actions. For the other two sectors sharing this intersection, if the small segment exists between the two sectors, then one must ask if there is a flow between the two; if there is, the edge should be orthogonal to the flow (or nearly so) and wider to allow for more variation on how aircraft end up crossing it.

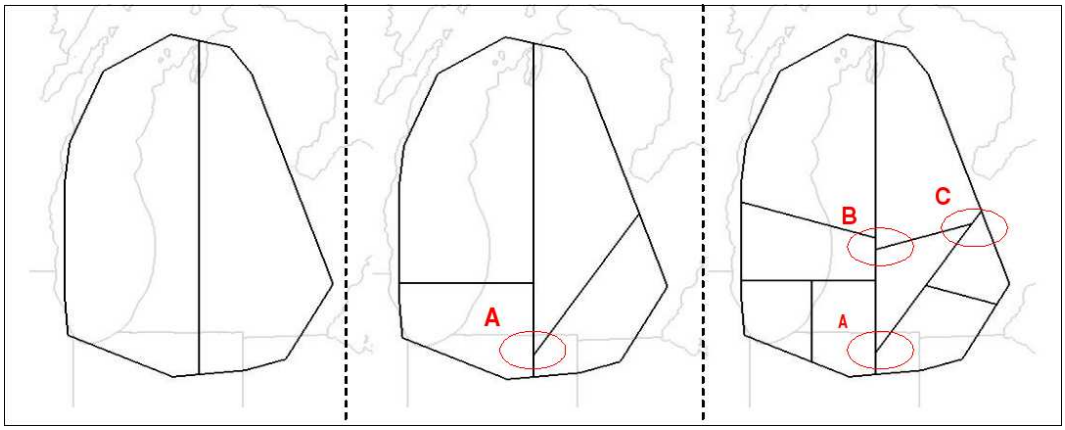


Figure 11: Problematic aspects of BSP Airspace Partitions.

In general, controllers felt that it was desirable to reduce the likelihood of a point out, which is when an adjacent sectors airspace is required in order to resolve a conflict. Thin sectors or sectors with acute angles have a higher probability of requiring point outs to resolve conflicts. Thus, acute angles should be blunted or cut off so there is enough space available to resolve conflicts. It was stated that there should be four or more sides per sector to avoid acute angle problems.

Pie-Cut Partitions Controller feedback indicated that, for most of the resulting sectors in Figure 12, sector size was acceptable, but that all sectors were highly undesirable due to the sector corners (nodes of the sectorization). Referring to the lettered regions in Figure 12, controllers provided the following comments:

Region D: There is an acute angle, which may mean that aircraft fly across the sector near the acute angle in a very short period of time. (This

comment depends, of course, on the aircraft traffic flows in the vicinity of the sector corner.)

Region E: The situation at this corner is even worse.

Region F: This corner is the worst. If a flow of aircraft were to pass close to this corner, they could move through five sectors in a matter of moments. High-degree corners should be avoided.

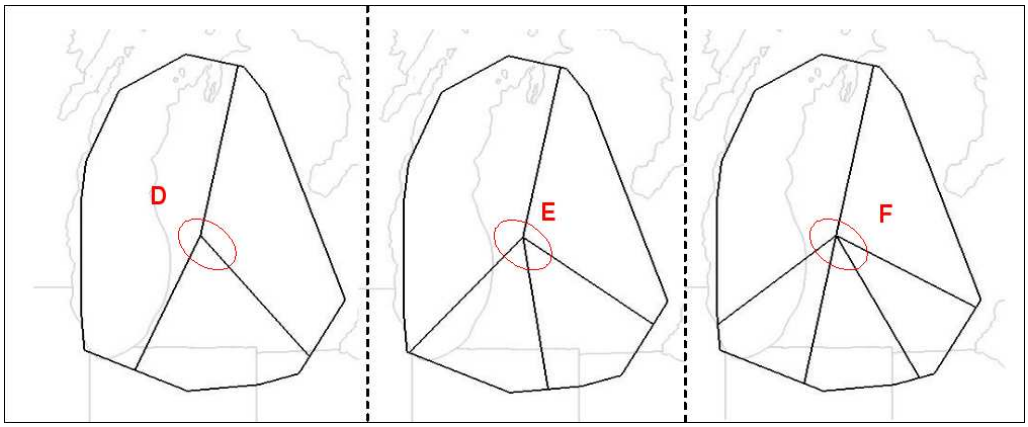


Figure 12: Problematic aspects of Pie-Cut Airspace Partitions.

In general, it was stated that, to avoid acute angles, the tips of the pie slices need to be blunted.

Wheel-Cut Partitions Controller feedback indicated that, for most of the sectors shown in Figure 13, size was acceptable, but the corners were problematic. Referring to the lettered regions in Figure 13, controllers provided the following comments:

Region G: To reduce the point out potential, blunt off the indicated tip shown in each of the three sectorizations in Figure 12.

Region H: The center wheel cut is good because it cuts off the acute angles on all the pie-shaped sectors in the previous example, but which is best depends on what you are trying to do with the flows. These sector boundaries best accommodate the case where flows were passing from the center to the perimeter sectors. If you have two flows the left case is

a good solution, if you have three then the center is a desirable solution, if you have four, then the right case is a good solution. You really need to know what the controllers are trying to do with flows in the very center sector in order to critique this further.

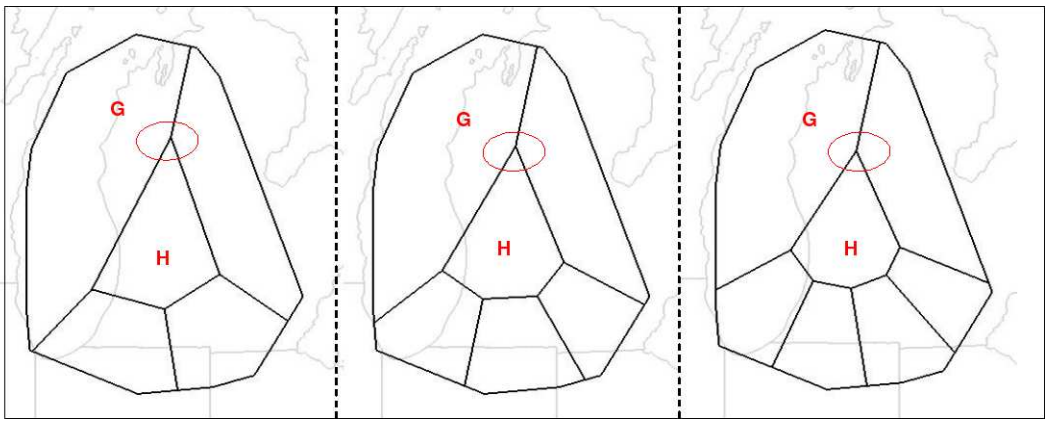


Figure 13: Problematic aspects of Wheel-Cut Airspace Partitions.

Further Feedback Controllers emphasized the importance of traffic flow considerations in sector design. Some specific feedback is shown in Figure 14. On the subject of aspect ratio constraints (from section 2.1.2), controllers felt that aspect ratio did not take into consideration two important points:

- If the flow is only in one direction and there is no crossing traffic, then a thin sector is acceptable.
- If there is any crossing traffic, then the width in the direction of that traffic needs to be long enough to resolve conflicts with other traffic. Note that the minimum width needs to take into account the speed of the crossing traffic; the constraint is not that the width needs to be a certain number of miles, but that the dwell time needs to be a certain number of minutes (typically a minimum of 4-5 minutes). Thus, the constraint must be a function of the average traffic velocity; the width will have to be greater, for example, if the crossing traffic is high altitude (faster) traffic, versus low altitude (slower) traffic. While controller feedback varied, it seems that a minimum width requirement should be approximately 20

nmi for low altitude traffic and approximately 30 nmi for high altitude traffic.

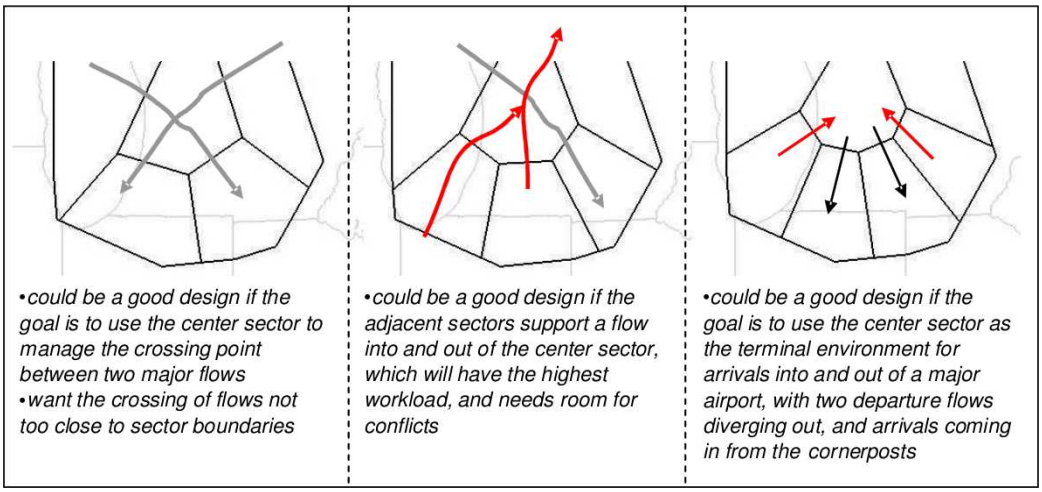


Figure 14: Influences of traffic flow characteristics on sector design.

Special Use Airspace (SUA), as the name suggests, is space assigned for special purposes like military usage, "no-fly" zones like Manhattan etc. More often it is defined as a region on the ground, with the airspace above it meant for special usage. There are various permanent SUA's all over the NAS and sometimes they are defined temporarily for certain amount of time, eg. a severe weather condition may be considered as a "no-fly" zone. Regardless of when and where they are present, the sector design should respect them in certain ways.

Ideally, the SUA region should be considerably inside the sector boundary. This gives sufficient time to the sector controller, after a flight has entered his sector, to re-direct it around the SUA, if needed. Sometimes a sector boundary may cut across the SUA, splitting it into two regions as shown in Figure 15. The requirement then, is that considerable part of the SUA should lie in both sectors. Thus both controllers are aware of its existence (and approximate size), and can direct the air-traffic around it. In other words, the sector boundary should not clip a corner of the SUA.

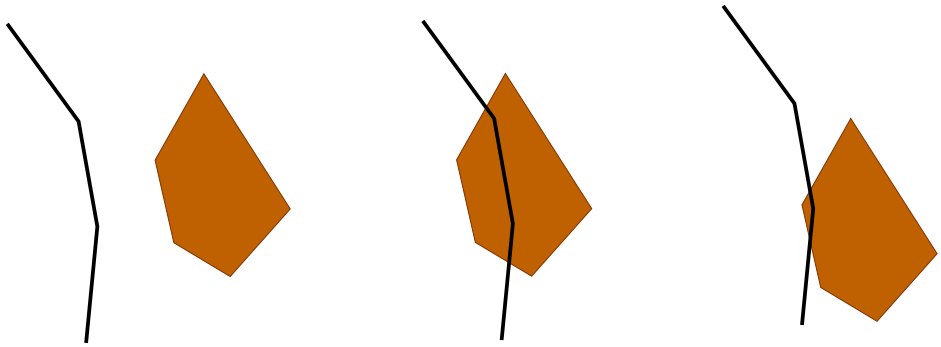


Figure 15: Desired interaction between the (brown) SUA and the (black) sector boundary. Left: SUA is considerably away from the sector boundary (desired); Middle: Sector boundary cuts through the SUA, with considerable part of SUA on each side of the cut (allowed); Right: Sector boundary clips a corner of SUA (disallowed).

2.3 Flow Conforming Sector Design

Most of the controller feedbacks dealt with how the sector boundaries should interact with the flows. Specific artifacts were also pointed out in the shape of the sectors, like small acute angles made at vertices of the boundary, more than three sectors meeting at a point, etc. Maintaining convex sector boundaries, obviously, becomes too restrictive to satisfy all the desired (shape and flow-interaction) properties, while at the same time balancing the workload across sectors.

Following the top-down paradigm of sectorization, we extend the notion of "cut", the partitioner at any level of recursion, from straight line segment to a more general polygonal path (with small number of bends). This allows more flexibility in the cut, thus making desirable features possible. Also we can now start with a non-convex region to sectorize. (Recall that earlier we started with the convex hull of the region, and at each step straight cuts guaranteed convextity of resulting sectors.) We start by modeling the controller feedback as specific constraints and implement them in our software extension GeoSect2.0.

2.3.1 Modeling Constraints

We model the feedback from controllers and the SUA knowledge in the form of constraints as enumerated below. Refer to Figure 16.

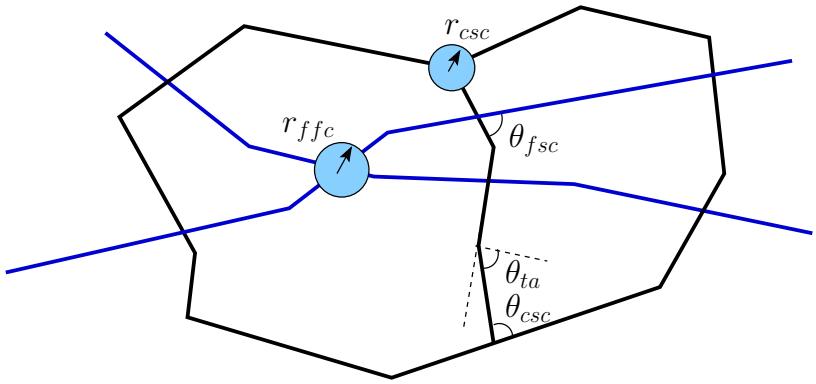


Figure 16: Desired features in a sector design modeled as constraints in GeoSect2.0

1. **Flow-Sector boundary crossing:** Dominant flows should cross a sector boundary almost orthogonally. We enforce this constraint by restricting the polygonal cuts which, if intersect dominant flows, have the (acute) intersection angle $> \theta_{fsc}$, $(0 \leq \theta_{fsc} \leq 90)$.

2. **Flow-Flow crossing:** In case of any crossing traffic, the width in the direction of the arriving traffic needs to be long enough (average dwell-time typically a minimum of 4-5 minutes) to resolve conflicts with other traffic. We model this requirement, by placing a rectangular obstacle of length l proportional to the average dwell-time along the arriving dominant flow, as shown in Figure 17. The width of this rectangle w is guided by the separation desired between flow and sector boundary. We also put a disc obstacle (of diameter w) at the crossing point, so that it remains sufficiently inside the sector. For now, the implementation only keeps a disc of radius r_{ffc} at the crossing point.

3. **Cut-Sector boundary intersections:** The angle made by a cut at the point where it meets the sector boundary should not be too acute, as the resulting sector will have sharp angle at one of the vertices. Like above, we enforce this angle $> \theta_{csc}$. Also, after we make a cut, we add

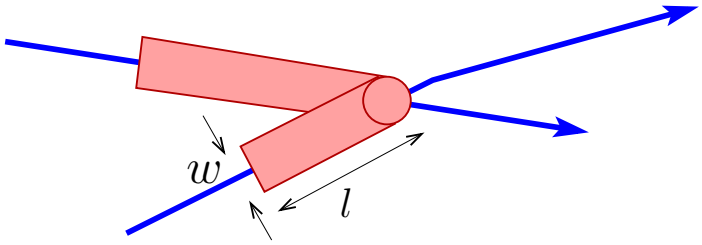


Figure 17: Directional dominant flows shown in 'blue' and the buffer from the sector boundary is modeled as rectangular and disc obstacles constraint.

a disc obstacle of radius r_{csc} at the point of intersection of cut and the region boundary. This not only avoids more than three sectors meeting at a point, but also ensures that no neighboring sectors share boundary of length $< r_{csc}$ (important for easy hand-offs).

4. **Turn-Angle at vertices along the polygonal cut:** The turn-angle made at an internal vertex of the polygonal cut should also not be too acute, for same reason as in constraint 3. We add a constraint: turn-angle $> \theta_{ta}$, $(0 \leq \theta_{ta} \leq 90)$.

5. **SUA-Cut interactions:** It is easy to incorporate the constraint which would keep the SUA considerably inside the sector boundary; by offsetting the sector boundary externally with the desired separation margin, and treating the new offseted polygon as a constraint, which all cuts must avoid. But, so as to allow a sector boundary to cut across the SUA (splitting the SUA, with considerable portions on both sides of the cut), we use an inner offsetting of the SUA as shown in Figure 18. The final constraint is, either the cut must completely avoid the external (outer) offseted polygon, or if it intersects the outer offset, then it must also intersect the inner offset. Along with reasonable values of the above angle constraints, this ensures that if the SUA is split, it is not just clipped. For simplicity in the current implementation, instead of using the offsets, we put disc obstacles of radius r_{sua} at all vertices of SUA boundary (or after boundary simplification, in order to have small number of vertices defining the sector boundary). This serves well for most cases.

6. **Sector area compared with the area of the convex-hull:** This constraint is intended to keep the sector shape close to convex. We

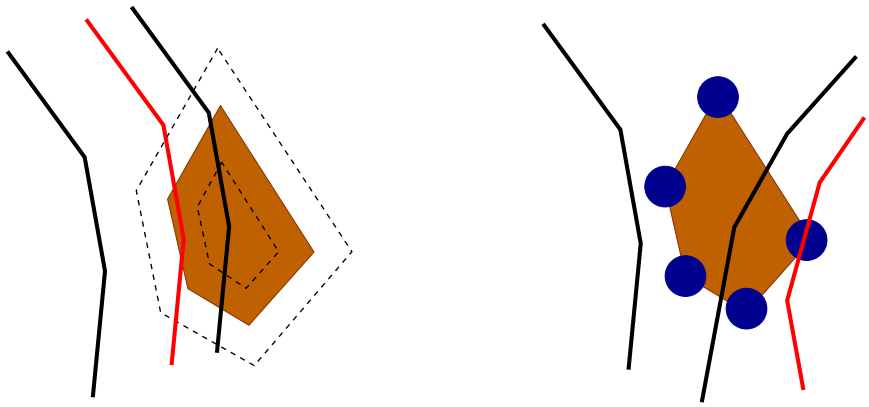


Figure 18: Modeling SUA Constraint: (black) sector boundaries are allowed while (red) boundaries are disallowed. Left: True modeling using inner and outer offset of the SUA boundary. Right: Simplified modeling using disc obstacles at the corners of SUA boundary, as implemented in GeoSect2.0.

constraint that the ratio of the area of sector region with the area of its convex hull should be $> \gamma$, $(0 \leq \gamma \leq 1)$. Again, with reasonable values for above angle constraints, this suffices to measure the convexity of sector i.e. a case like shown in Figure 19 is unlikely to happen.

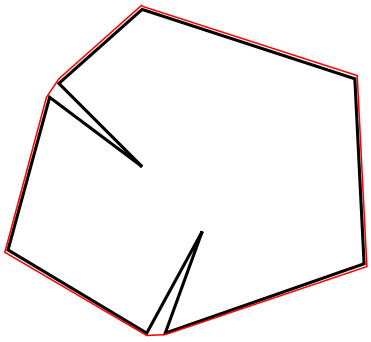


Figure 19: Bad (black) sector region and its (red) convex hull.

Cuts that satisfy the above angle constraints, and avoid any rectangular/disc obstacles, would result in sector designs that address the controller comments. The key question that remains is, how to ensure workload balance across sectors, while keeping them flow conforming.

2.3.2 Extending the Convex Sectorization Heuristics

The BSP heuristic (from Section 2.1.2) can now be extended as follows. At each step, find a polygonal workload balancing cut that satisfies all the constraints, and recurse until desired number of sectors are obtained (or the workload of each sector decreases below some threshold). Like before, it remains theoretically challenging to find an optimal workload balancing cut in 2D. Hence, we resort to finding "good" workload balancing cuts over a discrete search graph. See Figure 20, for an illustration of the steps of the BSP heuristic, for dividing ZDC into 8 sectors.

Discretizing the Search Space We first discretize the region boundary, by placing few (parameter dependent) points uniformly spaced along the boundary. Initialize the set of *boundary nodes* B with these points, and add to B the original vertices of the region. Only cuts that connect a pair of these boundary nodes will be considered at any BSP recursion step. Next, we discretize the interior of the region using a discrete uniform square grid and initialize the set of *internal nodes* I with these points. We augment, both B and I, with points that approximate the *medial axis* of the dominant flows. Start with a finer resolution (uniform) grid; the approximate medial axis points are the ones that are simultaneously close (minimum distance of a point from dominant flows is achieved for two flows at the same time) to two or more dominant flows.

An instance of the discretization explained above is shown in Figure 21. Each of the internal nodes can potentially act as a way point on the cut, giving the cut greater flexibility for good workload balance, while satisfying the model constraints.

We build a complete graph, $G(N, E)$, over the node set, $N = B \cup I$. From N, we remove all the nodes that lie in the vicinity (parameter dependent) of a dominant flow to avoid a cut from bending near a flow. Also, from E, we remove all the edges that violate any constraint (disc or angle). Note that, the turn-angle constraint depends on the previous edge selected in a cut, and the area constraint depends on the final partitioned polygons; hence these can not be checked up front, before actually searching for the cut.

Finding One Workload Balancing Cut After discretizing the search space, we wish to find a workload balancing cut from one boundary node to other. We start with a discrete set of c *allowable orientations* for our cut.

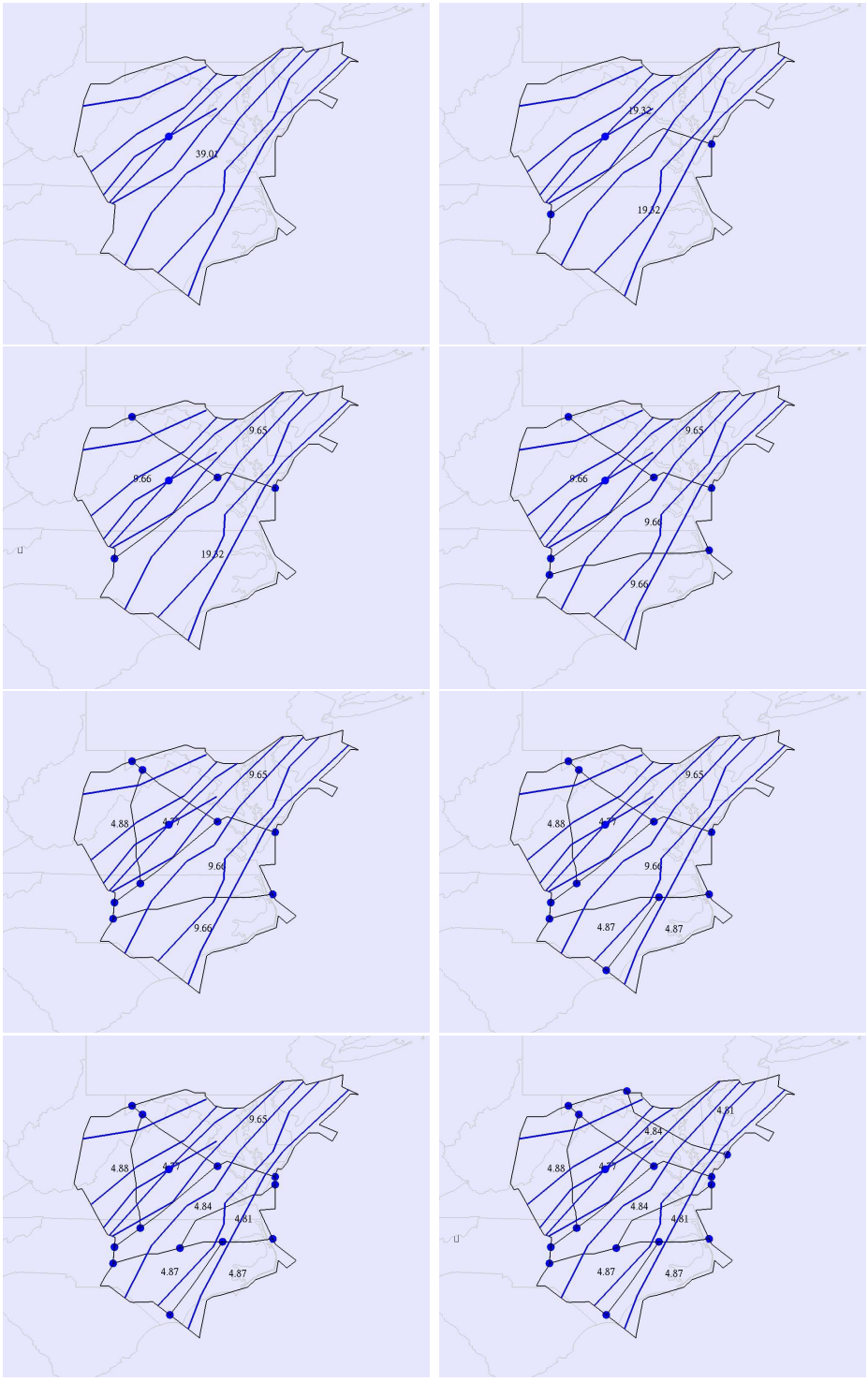


Figure 20: Steps of the BSP heuristic. ZDC partitioned into 8 sectors using hand extracted (blue) dominant flows. The numbers show time-avg. workload.

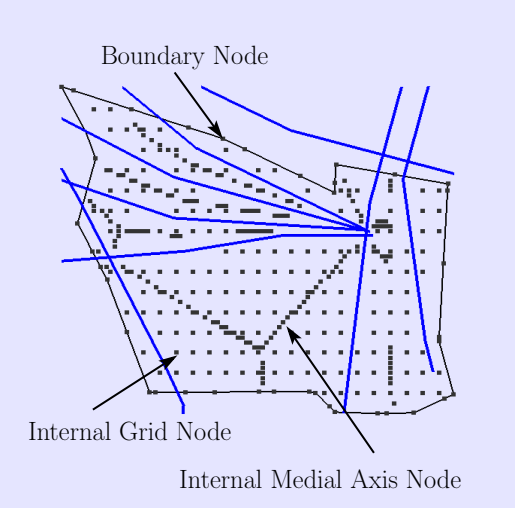


Figure 21: Nodes (points) generated by GeoSect2.0 for discretizing the search space with (blue) dominant flows.

For each orientation do the following. Consider all pairs of nodes, from B, such that the straight line joining them is parallel to the current orientation (within some allowable threshold). For each pair of nodes, $i, j \in B$, find a *min-turn-angle* depth first search path (refer Algorithm 1), satisfying turn angle constraint, starting from i to j. We use min-turn-angle path to keep the cuts as straight as possible, thus keeping the shape of sectors more or less convex. Note that (due to the way it is described), the mta_dfs path from i to j may differ from the path from j to i. Further, we explicitly check if the two polygons defined by this cut satisfy the area constraint (for approximate convexity). Finally, among all the constraint satisfying cuts, we pick the one that balances the workload best, over all orientations. In Figure 22, we show the workload balancing feasible cuts (in different orientations), for one recursive step of BSP. Each cut minimizes the maximum of time-average workload on two sides of the cut.

In Algorithm 1, the details of *computed_path*(i, j) are skipped, where in appropriate data structures are used to store the path computed by the depth first search.

Other heuristics, like Pie-Cut (and Wheel-Cut), can also be extended in a

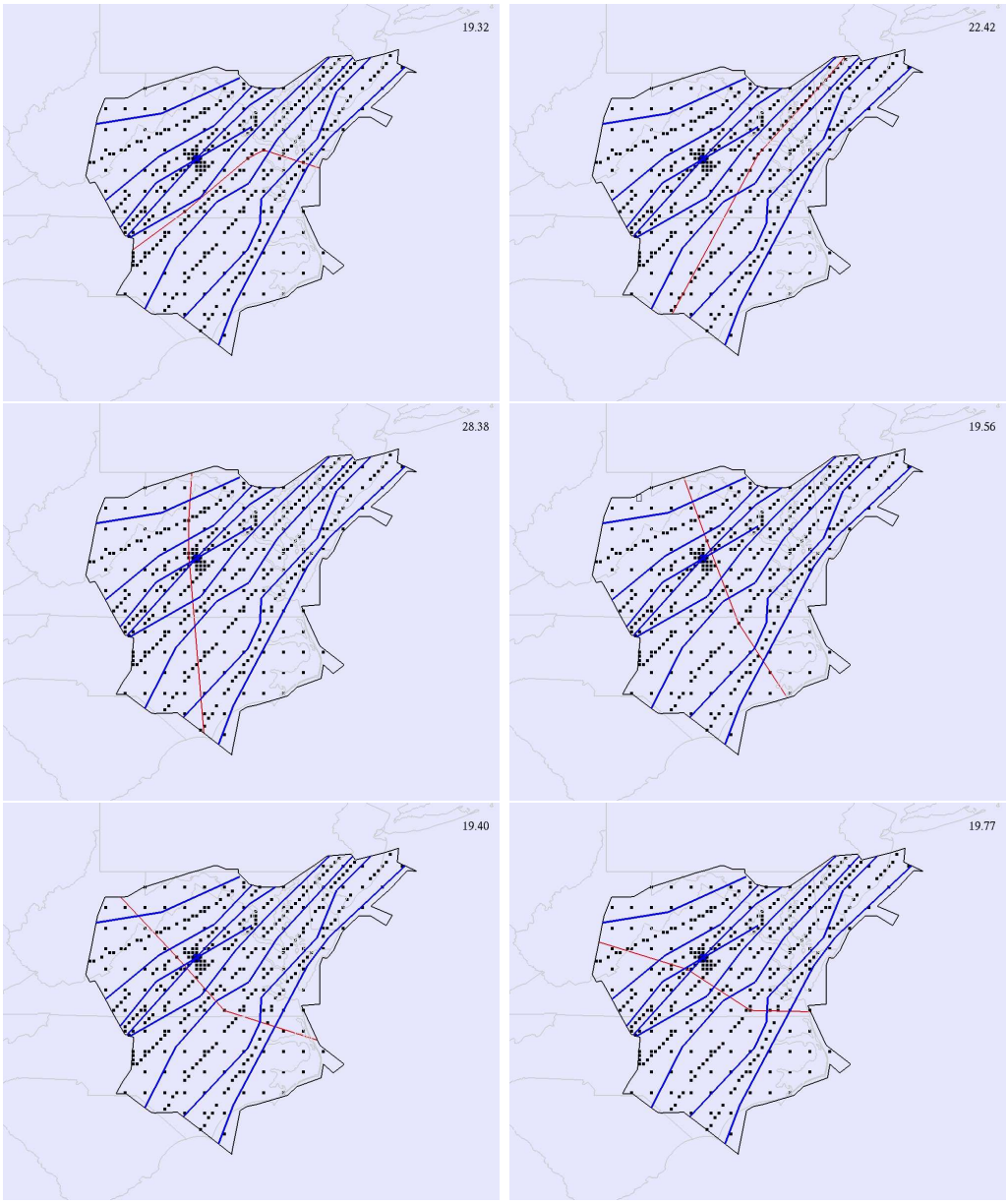


Figure 22: Feasible cuts in different orientations at one level of BSP recursion. Points inside the region are internal search nodes I and the number on top-right is the max time-avg workload.

Algorithm 1 mta_dfs

Input: Start node: i, End node: j, Graph $G(N, E)$, Set of boundary nodes B.
Output: Minimum turn angle depth first search path from i to j.
$T \leftarrow \{\}$ (Stack of nodes to visit)
$V \leftarrow B \backslash \{i, j\}$ (Set of visited nodes)
push T, i
while T is not empty **do**
  $u \leftarrow$ pop T
  $V \leftarrow V \cup \{u\}$
  if $u = j$ **then**
    return *computed_path*(i, j)
  end if
  $C \leftarrow \{\}$ (List of feasible current neighbors)
  for all nodes $v \in N$ **do**
    if $v \notin V$ and $e(u, v) \in E$ and $e(u, v)$ satisfies θ_{ta} **then**
      push C, v
    end if
  end for
  for all nodes $w \in C$, in decreasing order of $\angle wuj$ **do**
    push T, w
  end for
end while
return *null*

similar way. Instead of finding polygonal cuts connecting two boundary nodes, a central internal node may act as the pie-center, and polygonal cuts would connect this node to boundary nodes. We have not implemented these other heuristics in GeoSect2.0, yet. Next, we run experiments to analyze how various parameters and constraints affect the workload balance, and compare flow conforming sector designs for different data and objective functions.

We compute statistics for two new *flow-related* metrics, which quantify the "goodness" of sector boundary interactions with the dominant flows. The new methods for sectorization are bound to do good with respect to these metrics, since they have been encoded as specific constraints in the model.

1. **Angle of Intersection** (ϕ) of dominant flows with the sector boundary.
2. **Distance of Flow Crossing Points** (δ) from the sector boundary. In GeoSect2.0, (δ) is measured as Euclidean distance directly considering (latitude,longitude) for the coordinates of points (*flat* earth assumption). Hence unit of measurement is degree.

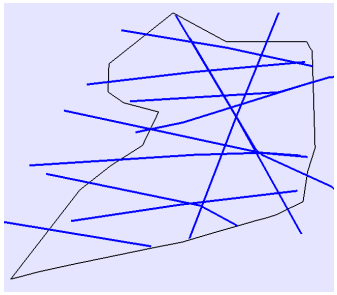
Note that, while [18] computes these metrics for all tracks, we restrict ourselves to evaluating these only with respect to the dominant flows.

2.3.3 Experiments

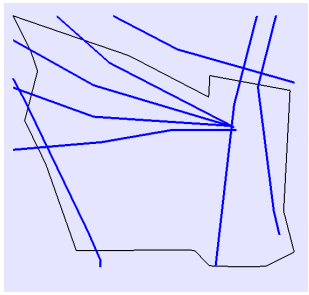
While code for GeoSect1.0 was written in C++, using Microsoft Visual Studio environment, all further versions (GeoSect2.0 onwards) have been developed under Linux (Debian) environment. The code still is in C++ (compiled with g++), and uses OpenGL (glut and glui) libraries for visualization. The experiments described below were run on a machine with Intel(R) Core(TM)2 Duo CPU (E4500 2.20GHz) and 4GB ram. (Some were run on a slightly better machine, with Intel(R) Core(TM)2 Quad CPU (Q9300 @ 2.50GHz) and 8GB ram).

Three datasets were used for most of the experiments. **Set1** (and **Set2**) consist of regions spanning 4 high-altitude sectors (currently operational in the NAS), from ZKC (and ZFW) center. Region for **Set3** comprises of the whole of ZDC center. The 452 tracks for **Set1** come from simulated data, and are meant to represent 90 minutes of high traffic time window. The 1327 tracks for **Set2** (6380 for **Set3**) are actual flown trajectories for a 24 hour time period. All sets have track data for altitude range $\geq 24k$. See Figure 23, for the

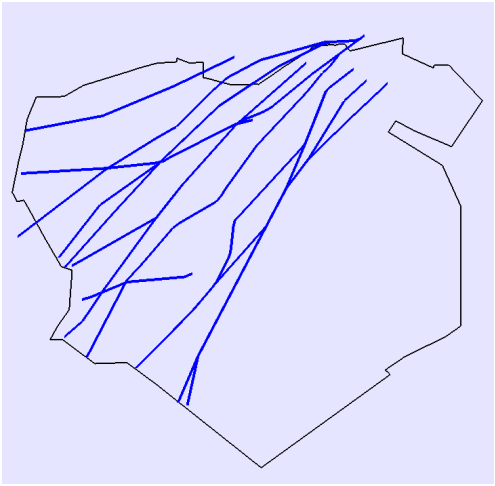
screenshots of the datasets. The dominant flows for all the sets are obtained by clustering the input trajectories, using methods described in Section 4.2.



(a) **Set1**.



(b) **Set2**.



(c) **Set3**.

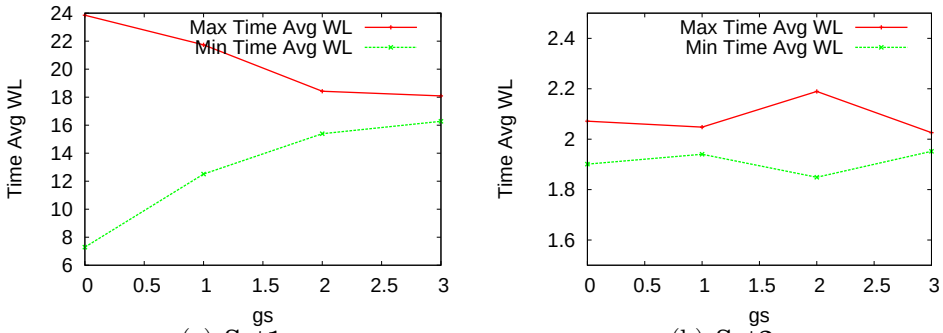
Figure 23: Datasets along with auto extracted (blue) dominant flows.

The first set of experiments include the sensitivity analysis of the workload balance to various parameters and constraints in the new model (mostly using **Set1** and **Set2**). After making intelligent choice for the parameters, experiments are conducted on **Set3** for different workload (objective) functions. For all experiments, at each step of BSP recursion, the region with maximum worstcase workload was picked for further sub-division.

Sensitivity Analysis For all sensitivity analysis experiments, the number of sectors desired for both **Set1** and **Set2** is 4. Also, the objective function is to balance the time average workload.

- **Search Space:** Size of the search space is dependent on the the number of nodes and edges in the search graph. There is a parameter gs which guides this size in GeoSect2.0. In particular, points are uniformly spaced along the boundary at a distance $total\_boundary\_length/(C \cdot 2^{gs})$ for some constant C. Similarly, the spacing between the internal (uniform square) grid points is $[0.5 \cdot (x_{span} + y_{span})]/(C \cdot 2^{gs})$ where $x_{span}(y_{span})$ is the x-width (y-width) of the region's bounding box in 2D. Thus, the number of search nodes at any level of recursion is proportional to $(C \cdot 2^{gs})^2$ i.e. with every increment in gs by 1 the number of search nodes becomes (approximately) 4 times. Parameter gs is kept in the exponent to ensure that for any specific value of gs, all search nodes coming from parameter values $gs' < gs$ are preserved.

  Note that, few of the uniform search nodes may be missing from the search space, either because they lie outside the region or near a constraint. Similarly, the edges violating any constraint may also be missing. Hence, increasing gs only guarantees an approximate (multiplicative) increase in the search space.



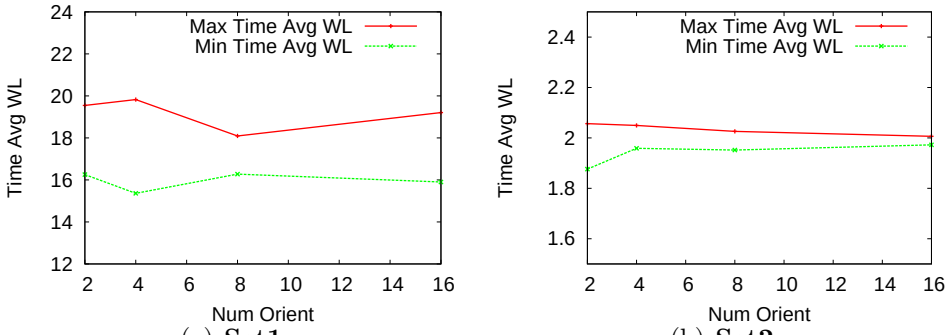
(a) **Set1**. (b) **Set2**.

Figure 24: Search space (gs) v/s time-avg. workload.

In Figure 24, we show the effect of increasing gs on workload balance. Not surprisingly, the workload balance improves by increasing the size of

search space, as number of choices for the cuts increases. For $gs \geq 3$ the workload balance almost becomes perfect. Also for **Set2**, good workload balance is already achieved for gs as less as 1. This shows that one may get lucky to find good search space for small values of gs. As discussed before, a higher value for gs is unlikely to hurt the workload balance, though it does have a huge impact on the running time. For **Set2**, as gs was increased from 0 to 3, the average number of search nodes per iteration (of recursive BSP) increased from 56 to 634, the average number of feasible cuts for all orientations (8 here) increased from 28 to 367 and the running time of the experiment increased from 0.033 minutes to 2.86 minutes.

- **Discrete Orientations of Cuts:** The graph in Figure 25 also verifies the intuition. The workload balance improves as the number of allowed orientations (hence the number of feasible cuts) increases. Note that beyond a point, when all pairs of boundary nodes are checked to find a cut, the increase in the number orientations would not increase the number of feasible cuts.

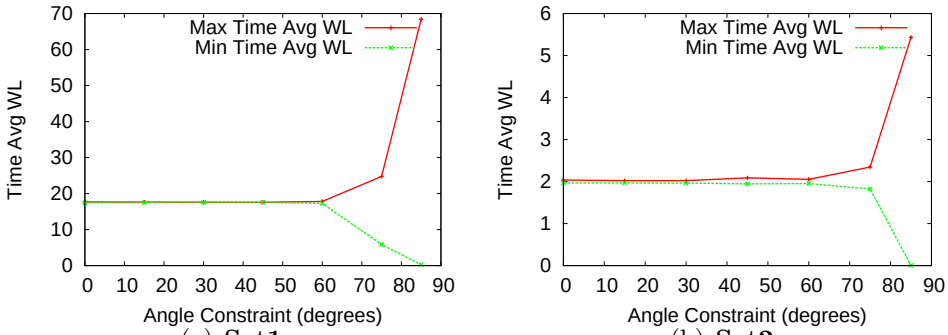


(a) **Set1**. (b) **Set2**.

Figure 25: No. of orientations (c) v/s time-avg. workload.

- **Model Constraints:** Various constraints in the model (as described in Section 2.3.1) can be broadly classified as angle constraints and disc constraints. The angle constraints θ_{fsc}, θ_{csc} and θ_{ta} all should ideally be close to orthogonal (90°). In Figure 26, we show the sensitivity of workload balance to the angle constraints, as they increase from 0° (no constraint) to 90° (max constraint). All the angle constraints are increased simultaneously as it is difficult (and may be unnecessary) to classify one as

more important than the other. The radius for all disc constraints were set to be 0 (no constraint). Very high values of angle constraint (75° and 85°) have a significant impact on the workload balance. In Figure 27, we show the sensitivity of average(over 4 sectors) minimum ϕ to the angle constraints. The average minimum ϕ consistently increases with the increase in the angle constraint.



(a) **Set1**. (b) **Set2**.

Figure 26: Angle constraint v/s time avg. workload.

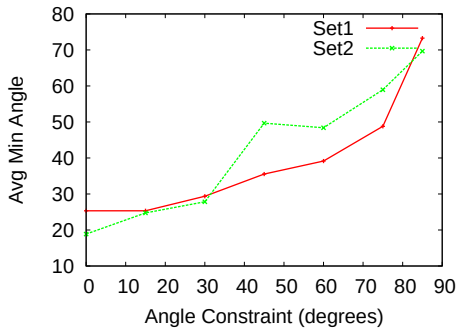
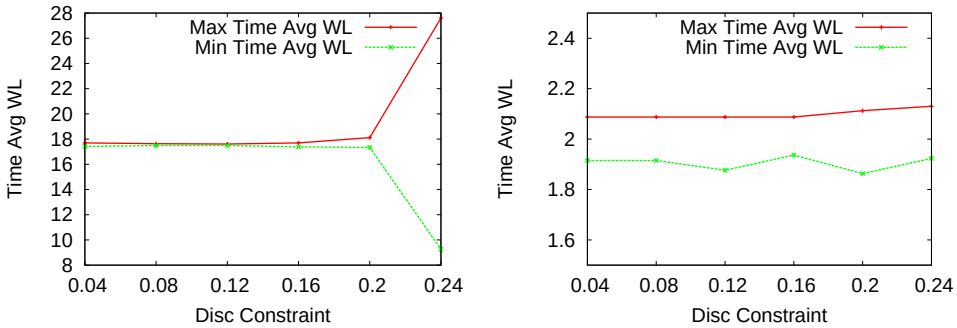


Figure 27: Angle constraint v/s avg. min ϕ.

The radius of two disc constraints: flow-flow intersection r_{ffc} and the cut-boundary intersection r_{csc} are also increased simultaneously and the effect on the workload balance is shown in Figure 28. The angle constraints were all set to be 0° (no constraint). Even for disc constraint as

high as 0.20 the workload balance is good. Note that, no SUA boundaries were used for any experiments; hence though the concept exists (and is implemented in GeoSect2.0), sensitivity of workload balance to the r_{sua} constraint is left as future work. In Figure 29, we show the sensitivity of average (over 4 sectors) minimum δ to the disc constraint. Similar to angle constraint, the average minimum δ consistently increases with the increase in disc constraint.



(a) **Set1**. (b) **Set2**.

Figure 28: Disc constraint v/s time avg. workload

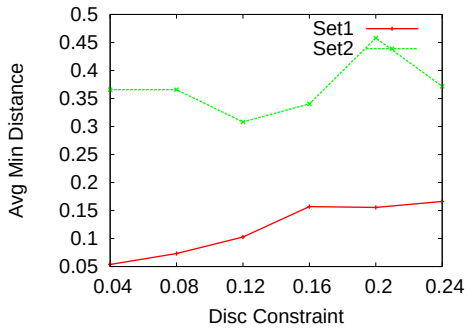


Figure 29: Disc constraint v/s avg. min δ.

Comparing different objective functions Using sensitivity analysis results, we pick reasonable values for each constraint, and experiment with **Set3**

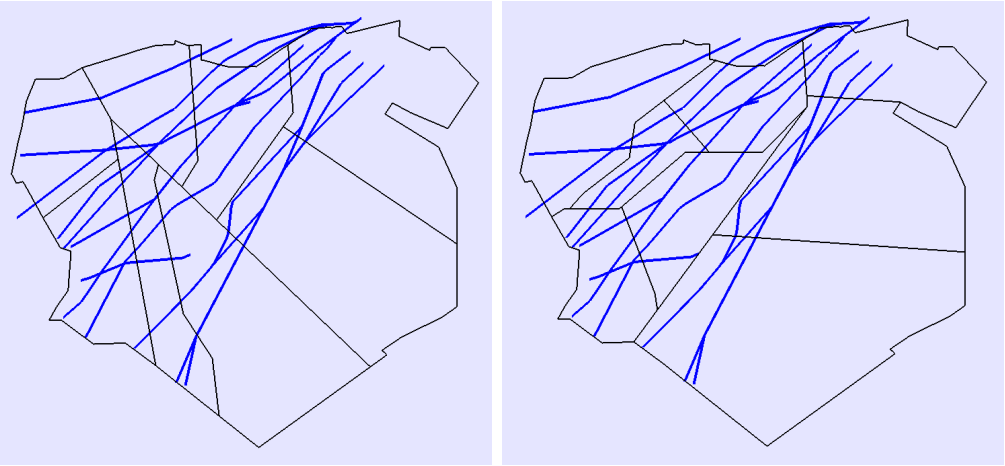
Sector Design	Time Avg WL		Avg. Dwell Time		Avg. Min. ϕ	Avg. Min. δ
balancing	Max	$Std.Dev.$	$Avg.$	Min		
Time Avg WL	13.88	0.09	8.81	5.17	65.85	0.39
Coord. WL	19.18	1.39	11.35	6.33	63.37	0.38

Table 5: Comparing workload and flow conforming metrics for two sector designs.

for different workload functions. In particular, we start by looking at two extreme cases, first experiment where we balance (minimize the maximum over all sectors) time average workload, and second where we balance coordination workload. A sector with low coordination workload has less intersections (and thus more alignment) with flows. Thus, balancing coordination workload favorably increases the average dwell time of air craft within the sector.

For this set of experiments, $gs = 1$ (this already gave sufficient graph size), the number of sectors was fixed to 8, all angle constraints were set to 45° and all disc constraints were set to 0.16. $c = 16$ orientations were used, so as to allow more flexibility to find cuts that align with flows. See Table 5, for the comparison of time average workload and average dwell time for the two experiments. As desired, first sector design gave excellent workload balance, but the average dwell time is considerably less. On the other hand, the second sector design successfully increased the average dwell time, though at the cost of time average workload balance. Both the designs performed very well on the average (over 8 sectors) minimum ϕ and the average minimum δ. Refer to Figure 30 for screenshots of the sector designs.

Next, we try to combine the objective functions from previous two experiments so as to get good workload balance, and at the same time try to align sector boundaries with the dominant flows. At each step of BSP recursion, among all feasible cuts, get the one with best coordination workload balance. Consider all the feasible cuts whose coordination workload balance is within $x\%$ of the best coordination workload. Among these, pick the one which gives the best time average workload balance. For $x = 0$, it would pick the best coordination workload balancing cut every time (like in second experiment above). The graph in Figure 31 shows the effect of increasing x on the average dwell time and the time average workload balance. It is interesting to see that as we relax the constraint (by increasing x) on requiring the cut to balance the coordination workload, we get improvement in workload balance.



(a) Balancing time average workload. (b) Balancing coordination workload.

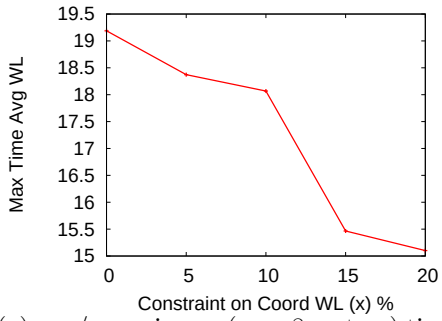
Figure 30: Results for **Set3**.

But, the monotonous decrease in the average dwell time is surprising, as there is no obvious reason for the two objective functions to compete each other. See Figure 32 for the screenshots of sector designs for the combined objective functions.

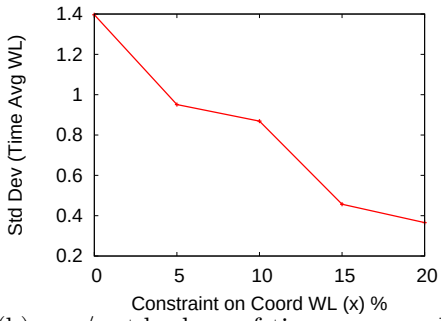
We also ran experiments for the entire NAS, considering each center separately for sectorization. Air traffic for a 24 hour time period was considered, and sector designs were computed for two different altitude layers: $24k - 35k$ and $35k - 60k$ feet. Hand extracted dominant flows (using traffic density heat map from Section 4.2) were used for all the centers. See Figure 33, for the screenshots. This experiment was run with an intermediate implementation of GeoSect2.0; hence there were a couple of constraints missing, leading to some visual artifacts in the sector boundaries.

2.4 3D Sectorization

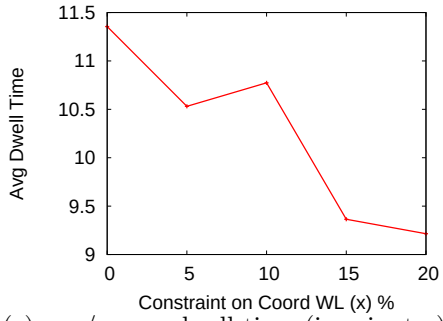
In all the above sections, we were trying to partition a region in 2D. A sector's workload was measured using all the air-traffic above (in the complete altitude-range) its polygonal boundary. This is restrictive in the sense that the altitude variation in traffic is not taken into account. Flows moving in opposite direction are always altitude separated. Also, depending on the air-



(a) x v/s maximum (over 8 sectors) time avg. workload.

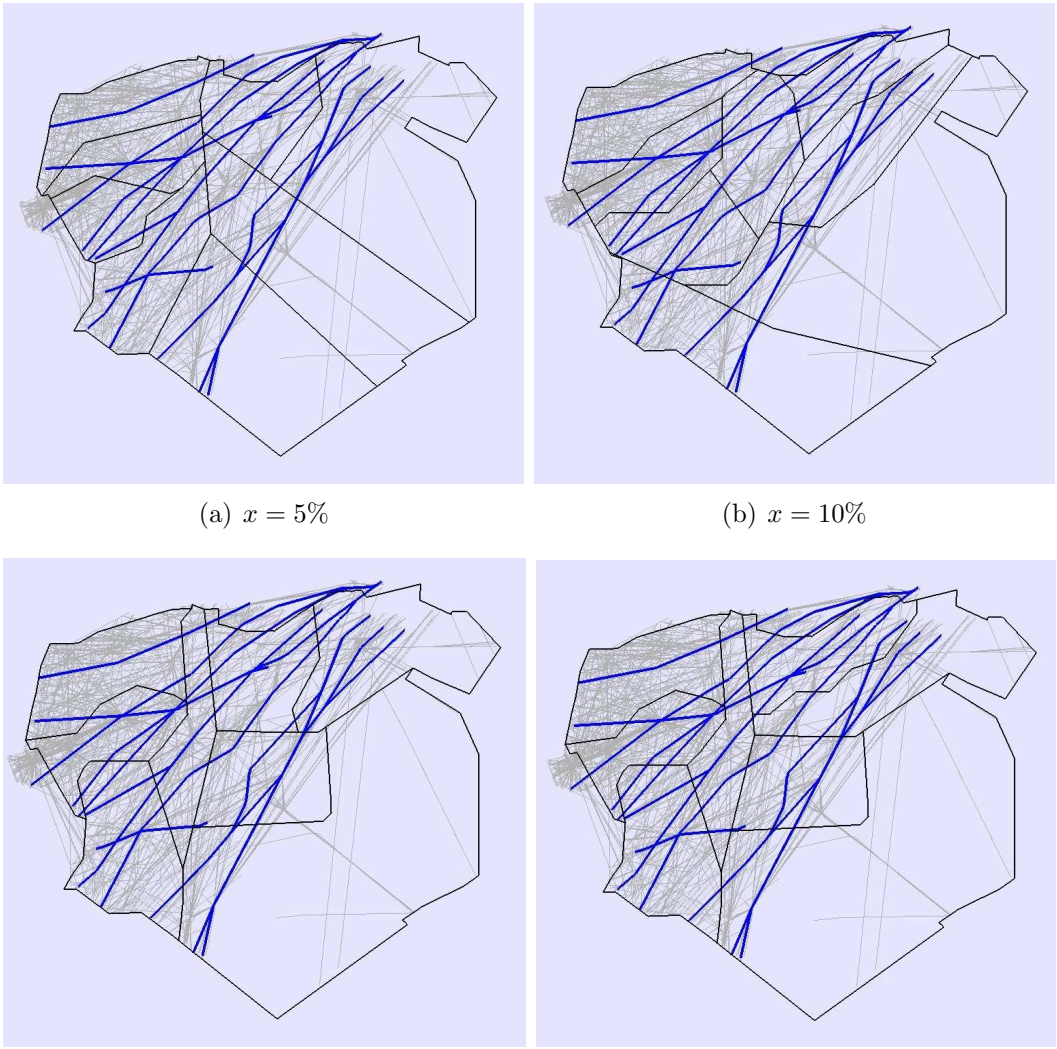


(b) x v/s std. dev. of time avg. workload.



(c) x v/s avg. dwell time (in minutes).

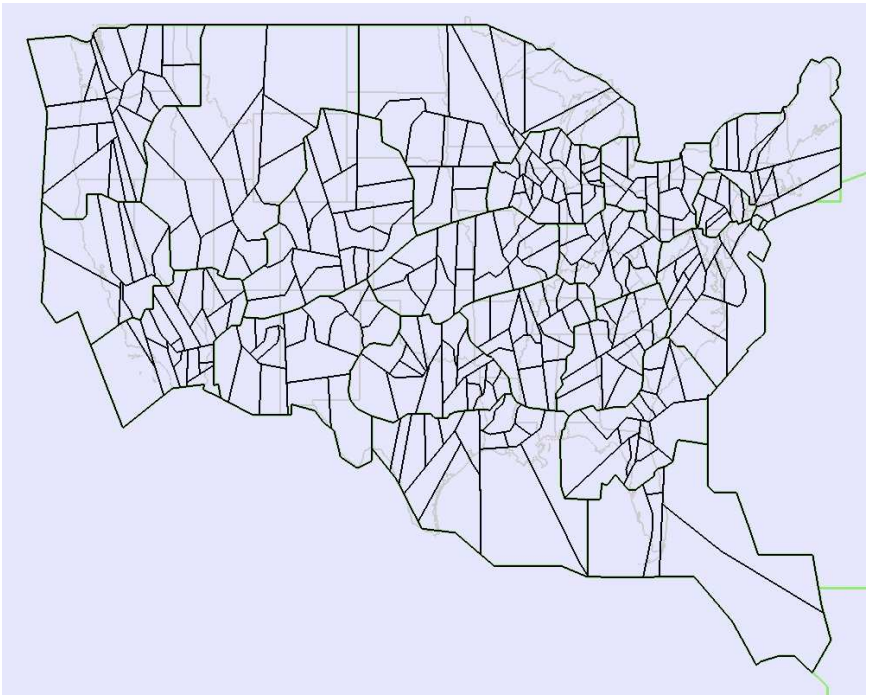
Figure 31: Effect of constraint (on the coordination workload balance) x.



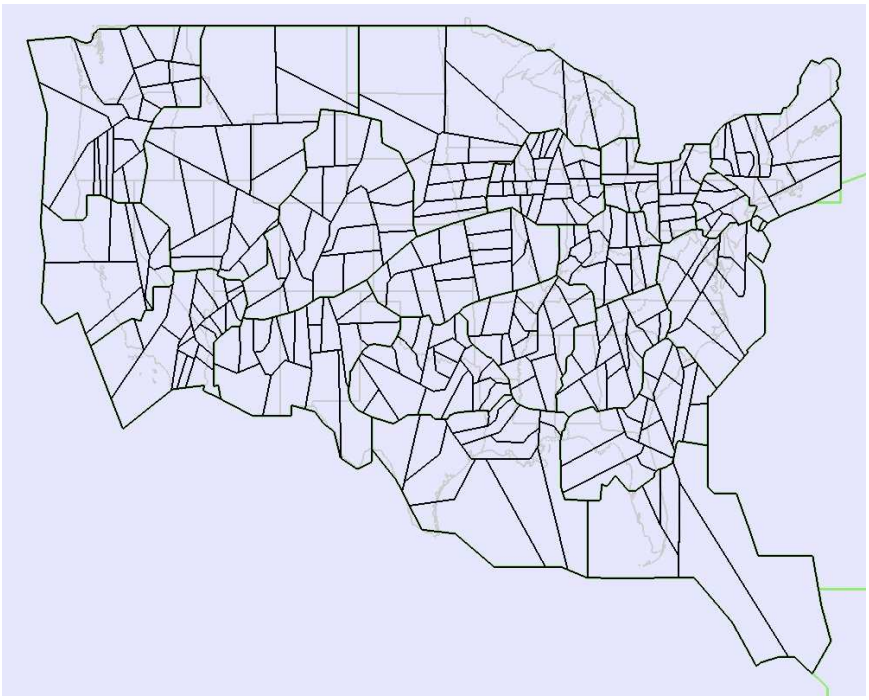
(a) $x = 5\%$ (b) $x = 10\%$

(c) $x = 15\%$ (d) $x = 20\%$

Figure 32: Screenshots of sector designs for the combined objective function.



(a) Alt. range 24k-35k feet.



(b) Alt. range 35k-60k feet.

Figure 33: Results for the entire NAS, one center at a time.

craft size, it may or may not have the ability to attain certain altitudes. Most important of all, flight ascends (during take off) and descends (during landing) make altitude consideration critical for sector designs in the TRACON region. Today's airspace in the NAS is also divided into low-altitude, high-altitude and very-high altitude sectors. Each existing sector thus has an altitude range (min and max) which filters the air-traffic handled by the controller in that sector.

Figure 34 gives an idea, how partitioning a sector at an altitude level might help reduce controller workload. Effectively it splits high altitude traffic from low altitude, thereby assigning workload of different altitudes to different controllers.

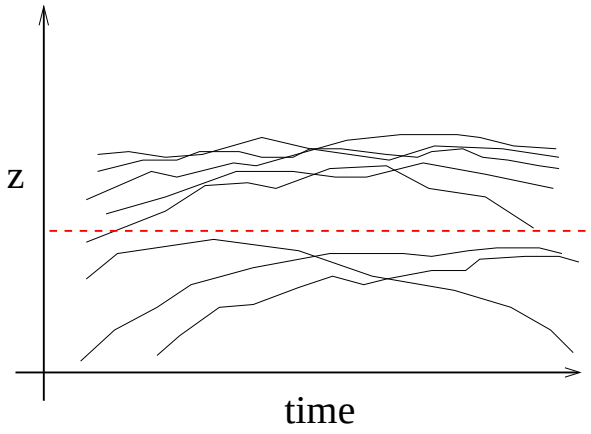


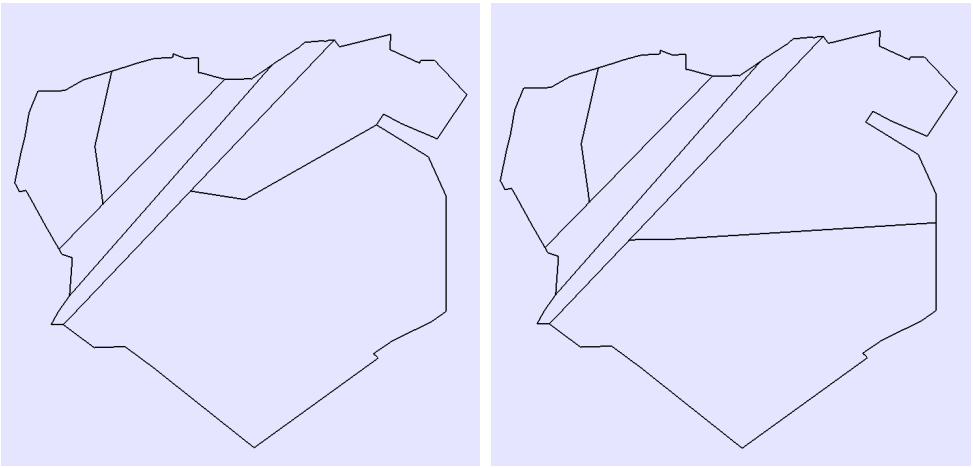
Figure 34: z=const. cut splitting high alt. traffic from low alt. traffic

Floors and Ceilings Intuitively, each sector has a flat *floor* and a flat *ceiling* defining its boundary in the z (altitude) dimension. Keeping the floor and ceiling flat is important. This is because of the limitations of the Operational Display Systems used by the controllers, which still gives them 2D visualization of the airspace. This is also true in case of sectors in the TRACON, where there is a lot of ascending and descending traffic. Hence, from a controller perspective it is easy to work with a specific range of altitude assigned to a 2D sector region.

It turns out that availing the possibility of this kind of partition (split) is an easy addition to the suite of heuristics GeoSect. The $1D$ problem (in

Section 2.1.1) was trying to find x = const. cuts; which is exactly what we want but this time in the z-dimension. The new heuristic for 3D sectorization (extension from BSP) is: At each level of BSP recursion, find the best z=const. cut (projecting the tracks in $z-t$ plane as shown in Figure 34). Compare the workload balance resulting from this cut to that resulting from a 2D BSP cut and pick the better of two. Note that one has to maintain an altitude range and use air traffic within that range for each intermediate region (and the final sectors).

Experimental Results We experiment with **Set3** (for 8 sectors balancing the worstcase workload), and compare the results for 2D and 3D sector designs. We do not use any dominant flows for this experiment. This is because, the current capability in GeoSect has flows defined only in 2D, and hence not all flows are relevant at all altitude levels. See Figure 35, for the screenshot of sectors produced by the 3D method. In particular, one intermediate region was divided at altitude level $34.5k$. This is not surprising, in fact it explains why the altitude split at $35k$ was suggested for the NAS wide experiment in Section 2.3.



(a) Alt. range 24k-34.5k feet. (b) Alt. range 34.5k feet and above.

Figure 35: Results for 3D sectorization of **Set3**.

Refer to Table 6, for comparison of workloads for 2D and 3D sector designs. Even though the 3D method marginally improved the max worstcase workload,

Sectorization	Worstcase WL			Time Avg WL		
	Max	Min	$Std.Dev.$	Max	Min	$Std.Dev.$
2D	37	33	0.42	14.30	12.36	0.22
3D	36	31	0.62	14.85	11.69	0.37

Table 6: Comparing results of 2D and 3D sector designs.

there is not much difference between the workload balance of the two methods. The 3D capability might be useful, when we model experiments for TRACON region, where there is a lot of ascending and descending traffic. Further, flow definitions need to be extended to 3D, and constraints in the model must also be updated appropriately, to make them suitable for 3D sectorization.

2.5 Conclusion

We started with convex sectorization methods and developed ways to balance workload, while keeping the sector shapes 'fat'. Further, we modeled dominant flow interactions as constraints and extended the methods for non-convex sectorizations, still keeping the shapes of sectors desirable. All the methods performed reasonable with respect to workload balance, and the non-convex sector designs gave good flow interactions. A method for 3D sectorization was also presented. In future, we would like to experiment more with other workload metrics. [56] is an ongoing work specifically aimed at incorporating simplified dynamic density (SDD)[38] metrics into GeoSect.

3 Dynamic Re-Sectorization

3.1 Introduction

In Chapter 2, we saw sectorization methods, where the sector boundary remains fixed for the entire time period, of the track data. Depending on the time of day, certain areas in the NAS have high air traffic, while others have low traffic. The traffic intensity across different regions is also affected by the presence of convective weather, change in the demand profiles at various airports etc. Hence, the workload across (static) sectors changes, over the day; See Figure 36. It is important to address the dynamic nature of air-traffic, and allow sector boundaries to morph to the changing traffic. Moreover, the sector boundary changes should be *less* frequent, as there is a "transition cost" associated with each modification. The transition cost may be due to the reassignment of ATC to the new sectors, the hand-offs of air craft from one ATC to other (as the sectors change), etc. A general rule of thumb is that the benefit of new design (improvement in workload balance, etc.) should sufficiently exceed the transition cost, in order to warrant a change.

Note that, in the dynamic re-sectorization problem, an initial sector configuration is given as input. This might be a configuration that is currently operational in the NAS, or computer generated for a reasonably small time window (3-4 hours) of air traffic (looking into the future). Below, we explore two methods: the first promotes doing "few" clean-sheet re-designs over the day, to balance the workload across sectors, while the second tries to identify dense traffic regions, as the air traffic evolves over the day, and restricts redesigns to only those regions.

3.2 Multiple "Clean Sheet" Sector Designs

Consider an extreme case, where the transition cost between sector designs is negligible and there is high benefit of reducing the workload variation (over the day) across sectors. A simple way to achieve this is to invoke GeoSect (or any other static sectorization tool), several times over the course of day, to

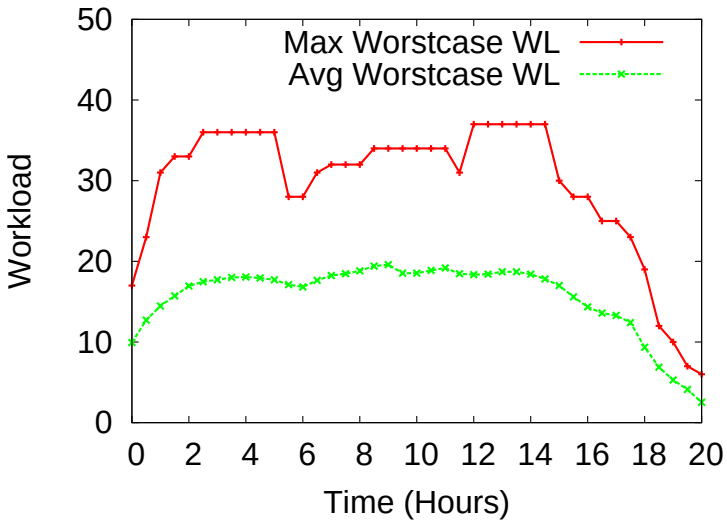


Figure 36: Time vs max and avg (over 17) worstcase WL for original sectors operational in ZDC. At each discrete time i, the air traffic in a 3 hour sliding time window $(i, i+3)$ was considered to evaluate the workloads.

do a "clean sheet" sectorization and balance the workload (as it exists at that time). If there is a bound on the maximum allowable workload of a sector, this method also decides the minimum number of sectors required, at each time of the day.

Such an extreme case, though, is highly impractical. A clean sheet redesign usually has a high transition cost associated with it; since (almost) all the controllers are reassigned to the new sectors. But, restricting one self to "few" redesigns (clean sheet), over the course of time T (eg. a day or a week), may be beneficial. If the redesign times are same as the shift-switching times of the ATC's, then arguably there is no transition cost. Also, if the phenomenon is repetitive and the ATC knows that at a certain (exact) time, every day, he is reassigned the same new airspace, its just a matter of time before he gets trained to handle two (or more) sectors. The sub-problem can be defined as: Suppose we are allowed to modify the sector design k times, over time T; identify the best switching times and the sectorizations for the resulting $(k+1)$ time windows.

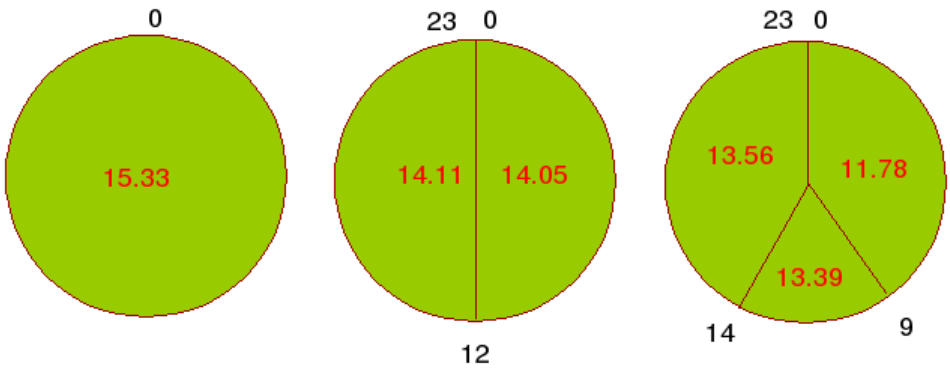
A discrete version of this problem, where all possible switching times come from a set of (pre-defined) discrete times, can be solved by a simple dynamic

programming approach. For each pair, (t_i, t_j), of time points (discretized at, say, 15-minute intervals), we use GeoSect to compute an optimal sectorization using a specified number of sectors. Then, over the time interval $[0, T]$, we compute an optimal set of k times t_i, at which to switch to a new sectorization that is optimal for the next time interval, (t_i, t_{i+1}) $((t_0 = 0)$.

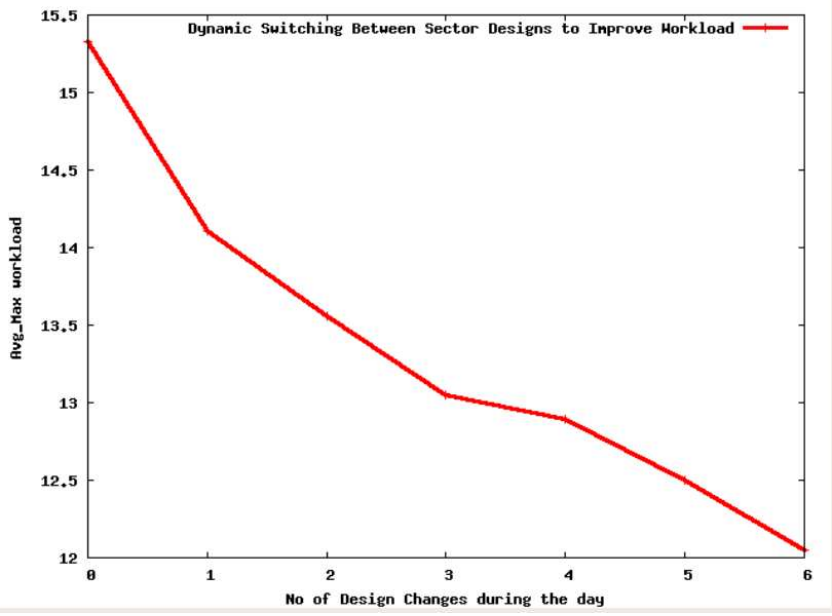
We experiment with Fort Worth (ZFW) center, using historical trajectories for a 24 hour time period. The number of desired sectors is 18, and the time is discretized into 24 intervals (hourly splits). We analyze the effect of number of switching times, k, to the max (over $k+1$ time windows) of average (over 18 sectors) worstcase workload. In Figure 37, we show how optimal re-sectorization can lead to a decrease in the maximum aircraft count. Refer to Figure 38, for the screenshots of resulting sector designs, for $t = 2$. Notice that, by allowing more switches (for the same number of sectors), we improve the max workload. Alternately, if we keep a bound on the workload, multiple switchings would help in decreasing the number of sectors.

3.3 Local Adaptable Re-Sectorization

Given a sectorization to start with (either the existing one, or the one generated by GeoSect), it is easy to identify high traffic regions (a group of local sectors) in the NAS. Locality is specified by means of a parameter, k: Within an existing airspace partition, we examine local neighbors (clusters) of k sectors (i.e., connected components of size k in the dual graph). Currently, GeoSect-D is implemented for the case $k = 2$. The objective is to perform small, local changes to the airspace, versus large-scale changes, in response to changes in traffic pattern and traffic density. This has several advantages over "clean-sheet" re-sectorization, including most notably cognitive familiarity. Local re-optimizations are also combinatorially small problems, which allows us to apply sophisticated optimizations that would be prohibitive if applied to global airspace design. The local re-optimizations within GeoSect-D can be done by any sectorization algorithm, such as those within GeoSect2.0, those based on MIP, those based on graph partitioning, Voronoi diagrams, etc. Currently, GeoSect-D is implemented only with re-optimizations based on the flow-conforming cut method (from Section 2.3).

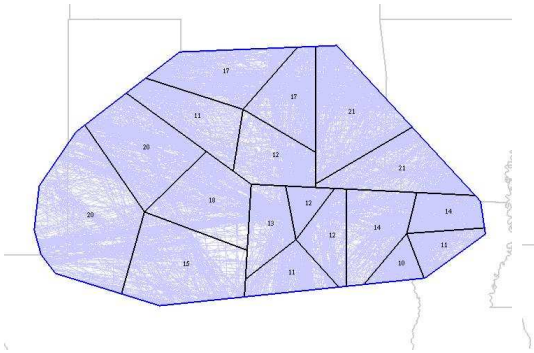


(a) Improvements to the avg. (over 18 sectors) peak workload for ZFW Center, as we allow k=1, or k=2, or k=3.

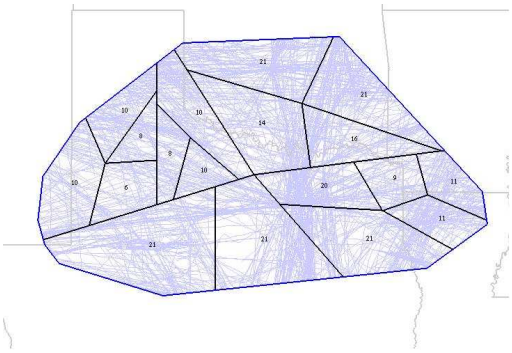


(b) Plot of max. (over all re-designs) avg. peak workload as a function of k.

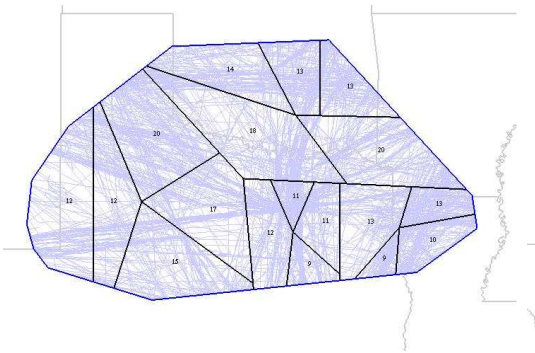
Figure 37: Effect of multiple (k) clean-sheet sectorizations, over the course of a day, on the peak workload.



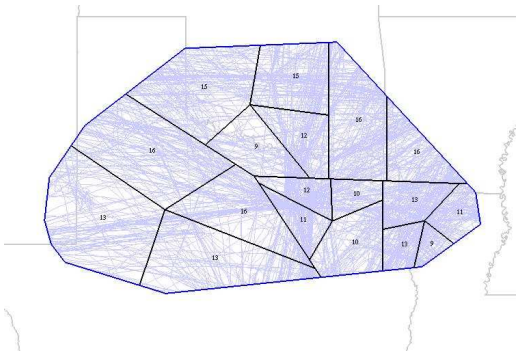
(a) 7:00 to 7:00 hrs (one sector design for the whole day).



(b) Design 1: 7:00 to 17:00 hrs.



(c) Design 2: 17:00 to 22:00 hrs.



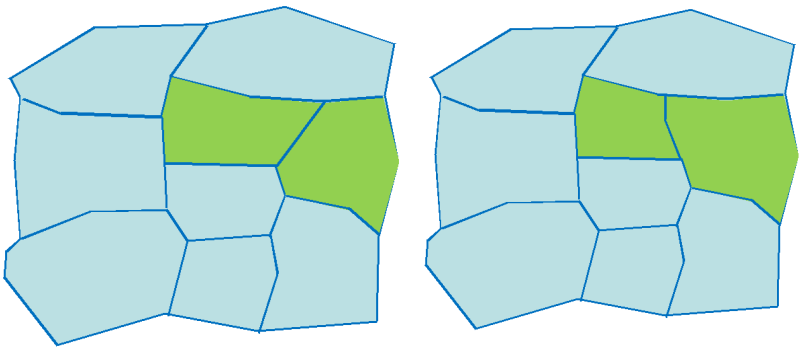
(d) Design 3: 22:00 to 7:00 hrs.

Figure 38: Results of the dynamic program to decide the best switching times (k=3) for sector design (using convex sectorization methods) in ZFW.

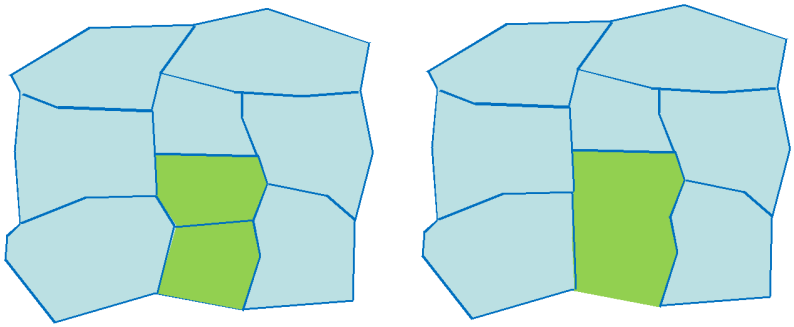
3.3.1 Algorithm

The main algorithm is a greedy, iterative method, which tracks over time the potential benefit to the objective function of performing each candidate local re-optimization. For the $k = 2$ case (the current implementation), this amounts to tracking each pair, (σ_1, σ_2), of adjacent sectors in the current design. We call this operation, a $2 \rightarrow 2$ move; see Figure 39 for an instance. We let $I(\sigma_1, \sigma_2)$ denote the amount by which the objective function improves if we perform a re-optimization of the pair (σ_1, σ_2) over the time horizon $[t, t + T]$, where t is the current time and T is the look-ahead time over which we examine the predicted demand. The objective function can be arbitrary and may be quite general, taking into account dynamic density or any workload metric that is computable. Simple versions of objective functions may just count maximum instantaneous aircraft count or average dwell time. More sophisticated objective functions may include several components that contribution to dynamic density, such as coordination workload, merging flows, altitude changes, speed variations among aircraft, etc. Since the local re-optimization searches over the set of feasible partitions of just the two sectors (σ_1, σ_2) (or, more generally, a small number, k, of sectors), it can afford to evaluate a complex objective function for each possible re-partition of $(\sigma_1 \cup \sigma_2)$. GeoSect-D reduces the space of all possible partitions of $(\sigma_1 \cup \sigma_2)$ by restricting itself to flow-conforming polygonal cuts computed by (static) sectorization methods in GeoSect.

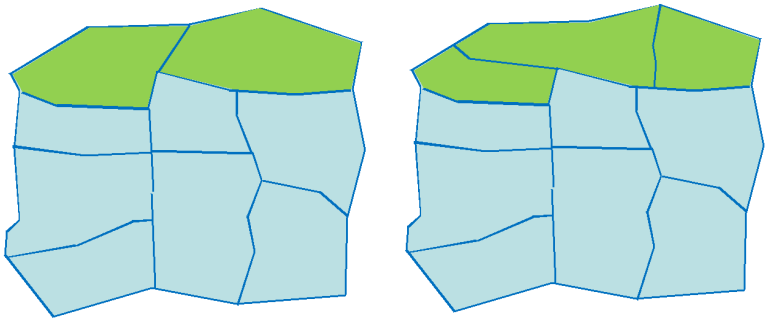
The GeoSect-D algorithm monitors the benefits functions, $I(\sigma_1, \sigma_2)$, over time. At each discrete time step, according to a re-evaluation interval of time (e.g., 5 minutes), the algorithm applies one of two possible trigger mechanisms: (1) We choose to re-optimize that pair (σ_1, σ_2) (or k-tuple) of sectors that corresponds to the largest benefit to the objective function, or (2) We choose to re-optimize all pairs that have benefit above a user-specified threshold. In both cases, we restrict ourselves to re-optimizing those pairs (σ_1, σ_2) that satisfy the condition that both σ_1 and σ_2 are eligible for re-optimization: We say that σ_i is eligible if it has been unmodified for at least a certain minimum amount of time, T_{min}, which specifies the minimum amount of time between changes to the boundary of a sector.



(a) $(2 \to 2)$ move.



(b) $(2 \to 1)$ move.



(c) $(2 \to 3)$ move.

Figure 39: Local moves available for adaptable re-sectorization.

3.3.2 Experiments

In Figure 40, we show a first experimental result with GeoSect-D applied to Washington (ZDC) center. The look-ahead time window is 3 hours long, and the re-evaluation time interval is 30 minutes. We plot the maximum aircraft count as a function of time, comparing the result for the original 17 sectors with the GeoSect-D result for 16 sectors (kept constant over time). We see that there is a reduction in the maximum aircraft count.

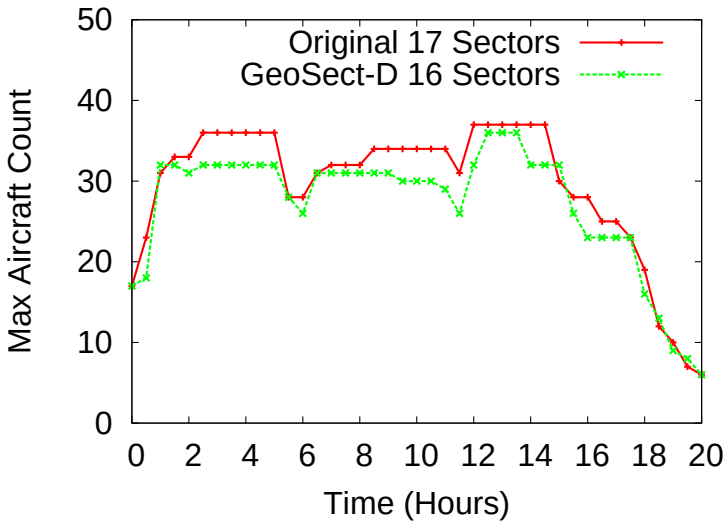
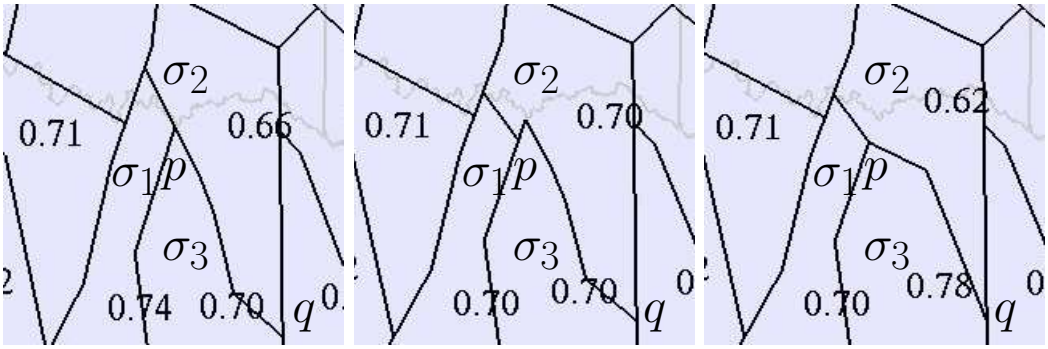


Figure 40: Plot of maximum aircraft count over time for ZDC, comparing the original 17 sectors to GeoSect-D results for a 16-sector design that is re-optimized over time using a 3-hour look-ahead and 30-minute re-evaluation interval

In Figure 41, we show an instance of $2 \rightarrow 2$ local move, as performed by GeoSect-D. Notice that, the local re-balancing between sectors σ_1 and σ_2 has resulted in an undesirable shape of the boundary (portion joining points p to q) between σ_3 and σ_2. The current implementation corrects this by finding a new cut, from p to q, which satisfies the shape/flow constraints. But, this shape correction results in worsening of the workload balance between σ_3 and σ_2. One idea to avoid this imbalance is, instead of finding a new path between p and q; we should search for a path from p to q', where q' lies any where on the boundary of either σ_3 or σ_2, and the cut p to q' balances the workload between σ_3 and σ_2. This idea is yet to be implemented in GeoSect-D.



(a) Detected unbalanced sectors σ_1 and σ_2.

(b) $2 \rightarrow 2$ local move balancing the workload between σ_1 and σ_2, results in undesirable boundary shape between σ_3 and σ_2.

(c) Correction to the undesirable boundary shape, results in workload imbalance between σ_3 and σ_2.

Figure 41: Stepping through the adaptable re-sectorization, as implemented in GeoSect-D.

The first experiment uses the assumption that the number of sectors is constant over time, which results from the fact that we do local re-optimizations of pairs of sectors, keeping the sector count fixed. In future experiments we will enrich the set of local operations to include $2 \rightarrow 1$, $1 \rightarrow 2$, $3 \rightarrow 3$, $3 \rightarrow 2$, and $2 \rightarrow 3$ re-optimizations (see Figure 39), thereby not necessarily conserving the number of sectors. This will allow the algorithm to adapt to changing demand by decreasing or increasing the number of sectors accordingly.

3.4 Conclusion

Dynamic re-sectorization is very important in DAC research. While improvements are possible with multiple clean sheet re-sectorizations, we believe that the local adaptable airspace re-optimizations are more practical, as they permit cognitive continuity over time of the airspace design. This leads to another interesting question: Given two sector designs, how close is one sector design to the other. A couple of approaches (shared-cell, Hausdorff distance and centroid-based metrics) are described in Bert et al. [30]. More importantly, what is the exact cost associated with the transition from one design to other.

It is also important to identify the lead time for the controllers, before a change in sector design actually happens, for them to prepare accordingly. Studies relevant to these questions using redesigns produced by GeoSect are underway.

4 Related Problems

4.1 Scheduling Aircraft to Reduce Controller Workload

As we have seen so far, ATC workload in a sector is largely related to the number of flights present in the sector at any given time, the more the flights the more is the work for the controllers. Departure time of flights thus affect the ATC workload because it decides when exactly the airplane is present in a sector. We study the problem of re-scheduling the departure time of flights so as to balance the workload and keep it as low as possible. This is joint work [36] with Joondong Kim and Alexander Kröller.

Problem Statement For a given set of trajectories and a given sectorization of airspace, determine alternate departure times "close" to the originally scheduled times so that the modified trajectories result in minimizing $\max_{\sigma,t} n_\sigma(t)$, the maximum occupancy count of a sector over a time window of interest.

Formally, the Min-Max Sector Workload Problem (MMSWP) is defined as follows: Given a set Σ of *sectors* and a set Θ of periodic *flight plans*. The common period of all plans is T, e.g., $T = 24$ hours. Corresponding to each flight plan θ is a sequence $\Sigma_\theta = (\sigma_{\theta,1}, \sigma_{\theta,2}, \ldots)$ of the sectors it visits, where $\sigma_{\theta,k} \in \Sigma \ \forall k$. Flight plan θ also has an associated *departure time* $d_\theta \in [0, T)$, and for each sector $\sigma_{\theta,k}$ it has an associated *dwell time* $t_{\theta,k}$.

Assuming a flight θ departs daily with a delay of Δ_θ, it will therefore be in sector $\sigma_{\theta,k}$ during the intervals

$$I_\theta(\sigma_{\theta,k}, \Delta_\theta) := [\sum_{\ell<k} t_{\theta,\ell}, \sum_{\ell\leq k} t_{\theta,\ell}) + d_\theta + \Delta_\theta + T\mathbb{Z} \tag{1}$$

Therefore, at time $t \in [0, T)$ (and also $t + zT$ for any $z \in \mathbb{Z}$), a total of

$$n_\sigma(t) := |\{\theta \in \Theta : t \in I_\theta(\sigma, \Delta_\theta)\}| \tag{2}$$

flights will be in sector $\sigma \in \Sigma$.

Our goal is to find delays $(\Delta_\theta)_{\theta\in\Theta}$ to minimize the overall maximum occupancy count $\max_{\sigma,t} n_\sigma(t)$. The delays are constrained to be within the range $[0, D]$ for parameter D. Note that additionally allowing flights to leave early, i.e., $\Delta_\theta < 0$, does not change the problem due to the periodicity of flight plans: A delay range $[-a, b]$ is equivalent to $[0, a+b]$, for $a, b > 0$. Therefore, we just consider the problem where $\Delta_\theta \geq 0$.

Relation to Job-Shop Scheduling When there is no constraint on the maximum delay, i.e., $D \geq T$, our problem is equivalent to "no-wait job-shop scheduling". We represent each flight plan as a job and each sector as a machine. We seek to minimize *makespan*, i.e., the smallest time in which all jobs can be processed, where no two jobs can be on the same machine at the same time. The no-wait constraint ensures that, once started, a job can neither be delayed between machines nor suspended while being processed on one. An optimal solution to the job-shop problem with makespan M can be converted trivially to a flight plan solution with maximum occupancy $\lceil M/T \rceil$. Vice versa, an algorithm for flight plan scheduling also solves job-shop by finding the largest λ for which a flight plan with all processing times scaled by λ can be scheduled with maximum occupancy 1. This can be achieved using binary search.

Lemma 4.1. *Minimizing makespan in the no-wait job-shop scheduling problem is polynomially equivalent to the Min-Max Sector Workload Problem (MMSWP).*

No-wait job-shop scheduling has attracted various researchers (see, e.g., [41, 51, 55, 50, 40]). [10] gives a *PTAS* for a special case of the problem and shows hardness of approximation for another case. [33] provides a survey of scheduling algorithms, defining the various terms and known results for some of the basic problems. Since the job-shop problem is NP-hard, so is the MMSWP, by Lemma 4.1. [23] formulates train scheduling as job shop problem with no-store constraints.

Another good reference is Bertsimas et. al [16] which solves an optimal combination of flow management actions, including ground holding, rerouting, speed control and airborne holding on a flight-by-flight basis.

Simplified Cases Here, we examine some special cases of the problem. In all the cases here, we consider $D = T$, so that there are no maximum delay constraints.

One-Sector Problem In the simplest of cases, there is only sector σ_0 and hence all the flight plans just define the time interval the flight remains in this sector. For all $\theta \in \Theta$, $\sigma_{\theta,1} = \sigma_0$.

If we remove periodicity of flight plans, i.e. put a constraint $d_\theta + \Delta_\theta + t_{\theta,1} \leq T$ hours for each flight θ, the optimal re-scheduling problem of minimizing the *max-workload* exactly maps to the bin-packing problem which is known to be hard (reduction from set partition) and also has an asymptotic PTAS [25][1].

If we consider periodic flight, then the *one-sector* problem has a trivial solution just by assigning delay to make flight back to back. This gives max-workload of $\lceil \sum_{\theta \in \Theta} t_{\theta,1}/T \rceil$.

Two-Sector Problem The extension of the problem to two sectors, with a periodic schedule of flights, seems like an interesting special case to understand the complications associated with the *no-wait* constraint and also the periodicity of the schedules. It is much easier to understand the *two-sector* problem by considering its exact equivalent below.

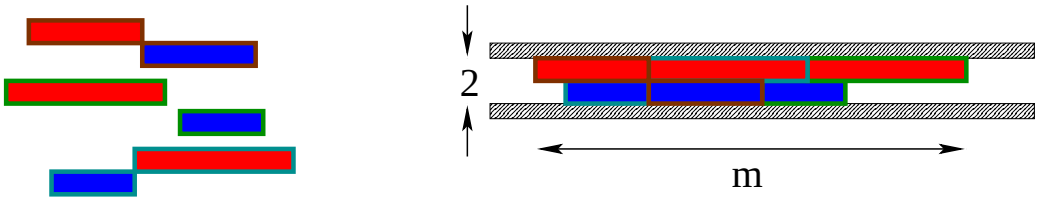


Figure 42: Left: 4 kinds of blocks. Right: The tight-fitting in the groove of size 2.

Consider Figure 42. Let A, B be the sectors. The red rectangles indicate the time interval of flights in A and the blue rectangles indicate intervals in B. Red to the left of blue indicates that flight starts in A and single red rectangle indicates the flight is only in A. Thus, the MMSWP corresponds to packing these blocks of rectangles as tightly as possible in the groove of width 2, constraining that red rectangles strictly remain in the upper row, blue rectangles

[1] An asymptotic PTAS is an algorithm that, given $\epsilon > 0$, produces a $(1+\epsilon)$- approximate solution provided $OPT > C(\epsilon)$ for some function C, and runs in time polynomial in n for every fixed ϵ.

strictly remain in the lower row and none of the rectangles overlap.

It turns out that periodicity does not really help for this case, as this version of the problem also turns out to be *NP-complete* by reduction from 3-*PARTITION PROBLEM*.

Theorem 4.2. *The MMSWP within* 2 *sectors is NP-Complete.*

Proof. $3m$ numbers $a_1, a_2, \ldots, a_{3m}$ are given for a 3-PARTITION PROBLEM instance P. All of these number are between $B/4$ and $B/2$, where mB is the total sum of $a_1, \ldots, a_{3m}$. We show the optimal solution of minimizing workload overall sectors gives us the solution of this problem.

Let's construct the MMSWP problem instance corresponding given input m, B, and a_i's. There are two sectors σ_1 and σ_2. Let time horizon T be $(mB + m)$. For given numbers a_i where $i \in \{1, \ldots, 3m\}$, we generate flights θ_i which visits only σ_1 with staying time a_i. i.e, $\Sigma_{\theta_i} = (\sigma_1)$ and $t_{\theta_i,1} = a_i$ for $i \in \{1, 2, \ldots, 3m\}$. And we prepare additional m flights $\theta_{3m+1}, \ldots, \theta_{3m+m}$ which visit σ_2 for time $(B + 1)$ and then σ_1 for 1. i.e, $\Sigma_{\theta_j} = (\sigma_2, \sigma_1)$ and $t_{\theta_j,1} = (B + 1), t_{\theta_j,2} = 1$ for $j \in \{3m + 1, \ldots, 3m + m\}$.

Then, we claim that if we minimize maximum workload overall sectors for this problem as 1, then we are able to solve given P.

In order to make workload as 1 for σ_2, we have to arrange $\theta_{3m+1}, \ldots, \theta_{3m+3}$ back-to-back like dark-gray blocks in Figure 43. Then there are m intervals with length B in σ_1. Now finding a placement of $\theta_1, \ldots, \theta_{3m}$ (light gray blocks in Figure 43) to make workload of σ_1 as 1 is finding a partition of $\{a_1, \ldots, a_{3m}\}$ such that each sum is exactly B.

□

4.1.1 Algorithms

We designed heuristic algorithms and compare them with a given flight plan and a lower bound.

Shifting Starting with the original flight schedule, we pick the sector with worst max-workload (in case of tie check each one of them), and look at the time interval where the max-workload is worse. All the flights present in the

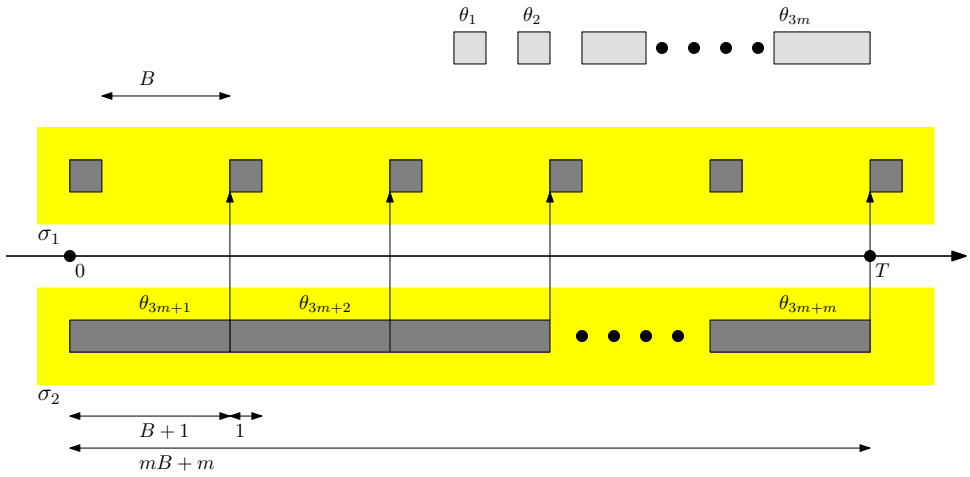


Figure 43: Constructing 2 sectors scheduling problem from a given instance of 3-Partition problem.

sector in that time interval are considered for re-scheduling (shifting) and the one which gives the "best" improvement is selected greedily. The goodness of a shift is judged by its effect on the workload vector which stores the workloads of all sectors in the sorted order. The flight whose re-scheduling gives the best improvement in lexicographic ordering of the workload vector is selected (in case of ties, we pick the flight which has the least difference in the re-schedule time and the ordinal schedule). The process is repeated till all shifts at a given iteration worsen the workload vector. (Note that shifts keep taking place even when the workload vector remains same).

We constrain the greedy shifting to be of the following three kinds:

- Right Shift - The flights are only allowed to be postponed.
- Left Shift - The flight are only allowed to be preponed.
- Short Shift - The decision of postpone/prepone is decided by the amount of shift, and the shorter one is picked.

We run the experiments for each of these separately, and also run an experiment where for each flight both left and right shifts are checked. We call that "Best Shift".

We also devise an incremental heuristic, in which flights are added one by one (in a random order). With each new flight addition, we run complete experiment of a shift heuristic considering all the flights previously added along with this one.

Randomized Rounding The *randomized rounding* algorithm solves a linear problem formulation whose variables describe a probability distribution for each flight plan. Then, a solution is generated by drawing delays from these distributions.

We evenly divide the interval $[0, D]$ into a discrete set of delays $\{0 = d_0, d_1, \ldots, d_m = D\}$. Also we slice the 24h-period T into n pieces $\{0 = t_0, t_1, \ldots, t_n = T\}$.

For each flight θ, the linear formulation has a variable $x_\theta(d_i)$ for each $d_i, 0 \leq i \leq m$. The interpretation (in terms of the finally assigned delay Δ_θ) is

$$x_\theta(d_i) = \Pr[\Delta_\theta \geq d_i] \; .$$

So the $x_\theta(\cdot)$ define a probability function on $[0, D]$ for every flight (the density is constant within each interval $[d_i, d_{i+1})$, that is, the distribution is uniform within each interval). To make sure the $x_\theta(d_i)$ define a proper probability distribution, we use the constraints

$$1 = x_\theta(d_0) \geq x_\theta(d_1) \geq \cdots \geq x_\theta(d_m) = 0.$$

This means the probability that a flight delay is in the range $[d_i, d_j]$ is $x_\theta(d_i) - x_\theta(d_j)$, so the probabilities are nicely encoded in the formulation. Note that

$$\Pr[\text{flight } \theta \text{ is in sector } \sigma \text{ at time } t]$$

is a linear term in the $x_\theta(\cdot)$ variables. To see this, translate t into a range $[\underline{\Delta}_\theta, \overline{\Delta}_\theta]$ of delays where a flight would start to be in σ at t. The probabilities are then:

- Some of the first interval with $d_i \leq \underline{\Delta}_\theta \leq d_{i+1}$, that is,

$$\Pr[\theta \text{ is in } \sigma \text{ at } t \; , \; \Delta_\theta \in [d_i, d_{i+1})] = \frac{d_{i+1} - \underline{\Delta}_\theta}{d_{i+1} - d_i}(x_\theta(d_i) - x_\theta(d_{i+1})) \; .$$

- All of the intervals $\underline{\Delta}_\theta \leq d_i \leq \ldots d_{i+1} \leq \overline{\Delta}_\theta$, in a similar fashion.

- Some interval part around $\overline{\Delta}_\theta$, again analogous to the first case.

By adding the cases, one can see how $\Pr[\theta \text{ is in } \sigma \text{ at } t]$ is a linear term with up to four coefficients. Obviously there are a number of special cases when $[\underline{\Delta}_\theta, \overline{\Delta}_\theta] \not\subseteq [0, D]$; these are easy to resolve and left out in this presentation. So we can now describe the expected load of sector σ at time t by the linear term

$$\text{E[number of flights in } \sigma \text{ at time } t] = \sum_{\theta \in \Theta} \Pr[\theta \text{ is in } \sigma \text{ at } t].$$

Hence, we solve the following LP:

$$\begin{aligned} \min\ & C \\ \text{s.t.}\ & \text{E[number of flights in } \sigma \text{ at time } t] \leq C \quad \forall \sigma \in \Sigma, t \in \{T_o, \ldots, T_n\} \\ & 1 = x_\theta(d_0) \geq x_\theta(d_1) \geq \cdots \geq x_\theta(d_m) = 0 \quad \forall \theta \in \Theta\ , \end{aligned}$$

which gives us a probability distribution for each Δ_θ, so we now generate actual Δ_θ values following these distributions.

An interesting variant arises when we add integrality constraints to the LP, as this forbids smearing flights over many delay intervals. As the resulting IPs are typically impossible to solve within reasonable time, we employ a different strategy: First, the LP-based heuristic is run. We identify the most crowded sectors, and add integrality constraints for tracks passing these sectors. At the same time, we vary n and m for different sectors and tracks, such that the crowded sectors get a more detailed formulation than the others.

4.1.2 Lower Bounds

Naive Approach The optimal one sector solution for a sector σ (refer section 4.1), for $D = T$, independent of any other sector, is a naive lower bound to its max-workload attained by any scheduling and for any D. Thus we can optimize each sector individually, and pick the maximum value over all sectors to serve as a lower bound to the workload attained by the optimal scheduling.

Linear Programming The second lower bound algorithm is based on the randomized rounding algorithm. Assume that all the $x_\theta(\cdot)$ are binary, i.e., 0 or 1 (see Section 4.1.1 for details). If now $x_\theta(d_i) - x_\theta(d_j) = 1$, then flight θ will have a delay $\Delta_\theta \in [d_i, d_j]$.

For a track $\theta \in \Theta$, a sector $\sigma \in \Sigma$ and a time t, we again compute the interval $[\underline{\Delta}_\theta, \overline{\Delta}_\theta]$ of delays for θ under which θ will be in σ at t. Then we determine the smallest $d_i \geq \underline{\Delta}_\theta$ and the largest $d_j \leq \overline{\Delta}_\theta]$. Then, when $x_\theta(d_i) - x_\theta(d_j) = 1$, the flight will be in σ at t. So define $g_\theta(\sigma, t) := x_\theta(d_i) - x_\theta(d_j)$.

The following IP charges 1 towards the maximum capacity C when a track is guaranteed to be in σ at t:

$$
\begin{array}{lll}
\min & C & \\
\text{s.t.} & \displaystyle\sum_{\theta \in \Theta} g_\theta(\sigma, t) \leq C & \forall \sigma \in \Sigma, t \in \{T_o, \ldots, T_n\} \\
& 1 = x_\theta(d_0) \geq x_\theta(d_1) \geq \cdots \geq x_\theta(d_m) = 0 & \forall \theta \in \Theta \\
& x_\theta(d_i) \in \{0, 1\} & \forall \theta \in \Theta, i = 0, \ldots, m
\end{array}
$$

The optimal solution to this IP is a lower bound to the original problem. For efficiency reasons, we do not solve this IP directly, but rather its LP relaxation, which is obtained by dropping the integrality constraint.

4.1.3 Experiments

We use real-world flight track data and sector data from the National Airspace System (NAS). The data, as shown in Table 7, is divided into 5 sets depending on the number of sectors. The *alt-range* defines the range of altitude for the air-traffic in the sectors. The high-altitude sectors typically have *alt-range* $24,000$ feet and above. **Set1**, **Set2** and **Set3** considers flight tracks for the entire 24 hour time period while **Set4** considers only the flights that overlap a 4 hour time window. Note that the flight times may start or end outside the 4 hour time window. Also, **Set4** includes all the sectors spanned by these flights, thus having high-altitude sectors, low-altitude sectors and some sectors from Canada as well.

Set5 (random data) consists of a 300×300 (unit nautical miles) square region divided into 16 sectors in the form of a square grid. 64 (uniform) random cities were generated such that 10% cities had weight 10, 15% had weight 5 and remaining had weight 1. 4994 random flights were generated between (weighted uniform) randomly chosen city pairs such that each city had probability of selection proportional to its weight. The departure-time of a flight was (uniform) randomly generated between $0 - 24$ hours. The speed of the

	No. of Sectors	Alt-Range	Flights	Time Window
Set1	5	$\geq$ 24k feet	1904	$0-24$ hrs
Set2	18	$\geq$ 24k feet	3063	$0-24$ hrs
Set3	57	≥ 0 feet	12123	$0-24$ hrs
Set4	1281	Different	11986	$14-18$ hrs
Set5	16	$\geq$ 24k feet	4994	$0-24$ hrs

Table 7: Summary of Data Sets used for experimentation.

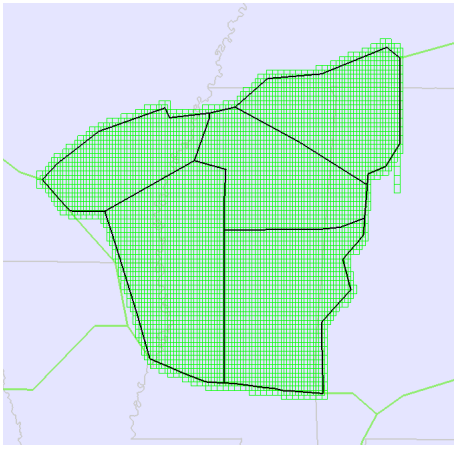
	Set1			Set2			Set3		
	Max	Mean	Var	Max	Mean	Var	Max	Mean	Var
Original plan	22	18.00	6.80	18	12.83	12.25	38	21.56	36.70
Right Shift	18	16.40	1.04	14	11.11	3.99	31	20.77	26.27
Incr. Right Shift	15	13.80	0.96	12	10.17	2.25	26	18.75	16.40
Rand. Rounding	16	14.20	1.36	14	11.17	3.69	29	20.18	21.55
MIP	15	14.20	0.16	14	11.33	3.22	27	19.68	17.71
	Naive	LP	IP	Naive	LP	IP	Naive	LP	IP
Lower Bound	6	9	10	5	8	9	16	20	wait

Table 8: Workload statistics of algorithms for Set1, Set2 and Set3. Max: Maximum Workload, Mean: Mean of workload, Var: Variance of workload

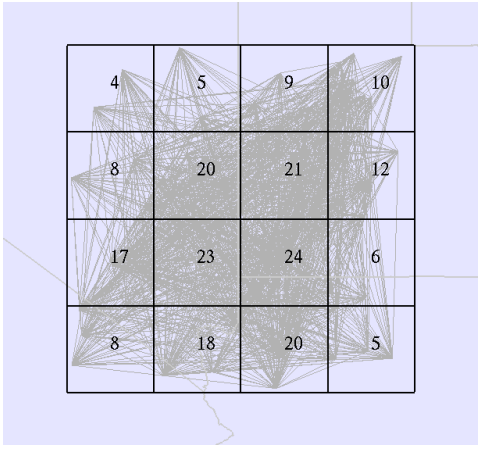
aircraft was modeled as a (uniform) random variable between $200-600$ nautical miles per hour. The arrival-time of a flight was thus calculated using the departure time, the speed of the aircraft and the distance between the city pair. Additional constraint was added that no two aircraft depart from (or arrive) at a city within 1 minute of each other.

A screenshot of data sets **Set1**, **Set2** and **Set5** can be seen in Figure 44.

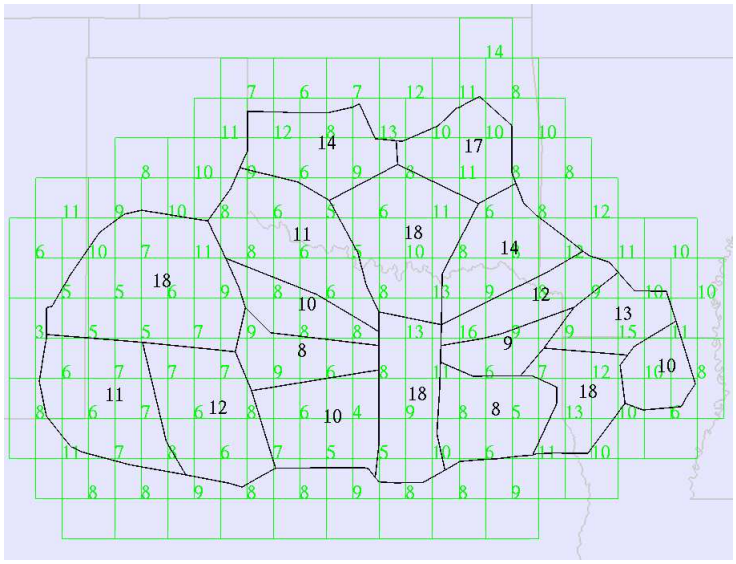
Tables 8 and 9 show the comparison of max-workload statistics of the given flight plans, the heuristic solutions and the LP based methods. The maximum allowable shift to any flight schedule was constrained to be 1 hour in all methods. The discretization of time for LP/IP methods is 1 minute. The results show a considerable improvement over the workloads of each sector arising due to the original flight schedules. Even the variance values have gone down significantly, indicating more balance of workload across sectors. In particular, the incremental shift heuristic seems to out-perform all the other methods. Note that the shifting heuristics do not discretize the time like LP/MIP methods.



(a) Set1 sectors and the underlying square grid (and shifted square grid) cover (grid resolution: 0.1x0.1).



(b) Set5 (randomly generated) flight tracks with the underlying sectors.



(c) Set2 sectors and grid cover (1x1).

Figure 44: Screenshots of datasets. The number in the sectors indicate the max-workload count for the corresponding flight schedules.

	Set4			Set5		
	Max	Mean	Var	Max	Mean	Var
Original plan	58	7.67	37.88	24	13.00	46.13
Right Shift	47	7.61	36.35	19	11.75	29.01
Incr. Right Shift	39	7.51	34.50	17	10.81	20.66
Rand. Rounding	39	missed	missed	wait	wait	wait
MIP	n/a	-	-	wait	wait	wait
	Naive	LP	IP	Naive	LP	IP
Lower Bound	12	-	-	13	11	11

Table 9: Workload statistics of algorithms for Set4 and Set5. Max: Maximum Workload, Mean: Mean of workload, Var: Variance of workload

	Set1 (1904 flt)			Set2 (3063 flt)			Set3 (12123 flt)		
	Max	Total	Avg	Max	Total	Avg	Max	Total	Avg
Right Shift	5.76	46.08	1.44	8.64	5:25.44	1.44	17.28	5:18.24	1.44
Incr. Right Shift	48.96	2:00:46.08	4.32	51.84	3:16:20.64	5.76	60	18:21:7.2	5.76
Rand. Rounding	60	13:22:23.52	10.08	60	13:06:48.04	5.76	60	35:18:15.4	4.32
MIP	60	14:21:48.04	11.52	60	15:21:42.28	7.2	60	37:10:59.08	4.32

Table 10: Time shift statistics of various methods for Set1, Set2 and Set3. Max: Max shift, Total: Sum of absolute value of shift, Avg: Average of absolute value of non-zero shifts. (format 14:21:48.04 means 14 days 21 hours 48.04 minutes)

The "missed" values in Table 9 and Table 11 is due to the failure of logging the workloads of all sectors. Since it takes about a week to run **Set4** with randomized rounding method, we did not have another chance to get those results in time. The "wait" indicates that we are still waiting for the instance to complete execution while "-" indicates that we do not hope to get the result for that case. For **Set4**, the MIP method ran for about 6 weeks and then ran out of memory, while the shift heuristic takes couple of hours and the incremental shift heuristic takes little more than a day to execute.

Tables 8 and 9 also shows the lower bound calculations for the 5 sets. The best solutions are still not close to the computed lower bounds but we believe they are very close to optimal solution. Future work will specifically aim to improve the lower bounds.

Tables 10 and 11 shows the statistics of the amount of time shifts from the original schedule. *Max* indicates the maximum shift in any flight schedule, *Total* indicates the sum of absolute values of shifts and the *Avg* gives the aver-

	Set4 (11986 flt)			Set5 (4994 flt)		
	Max	Total	Avg	Max	Total	Avg
Right Shift	53.28	12:53.28	4.32	6.97	3:8.32	0.82
Incr. Right Shift	60	14:22:53.76	17.28	53.56	4:18:4.84	3.84
Rand. Rounding	60	missed	11.52	wait	wait	wait
MIP	-	-	-	wait	wait	wait

Table 11: Time shift statistics of various methods for Set4 and Set5. Max: Max shift, Total: Sum of absolute value of shift, Avg: Average of absolute value of non-zero shifts. (format 14:21:48.04 means 14 days 21 hours 48.04 minutes)

age time shift of all flights with non-zero shifts. The value of *Total* in case of right shift heuristic is considerably small compared to other methods possibly because of the early termination due to local minimum. Also, the average time shift is considerably low for all the methods, thus suggesting that we can get considerable improvements in workloads with reasonable modification to the schedules.

Other Workload Considerations Apart from the *max-workload* of a sector, there are other workload issues which are significant from the controller perspective. One of them, usually referred to as *coordination* workload, deals with the hand-offs between controllers when an aircraft moves from one sector to the other. Another, critical one, is the *conflict resolution* workload which is related to monitoring the aircraft when they are expected to be present at (or near) the same (lat,long) point (conflict point) simultaneously. Note that even if two aircraft are flying at different altitudes, at the conflict point, they demand special attention of the controller.

While re-scheduling flights has no effect on the *coordination* workload, it can favorably affect the *conflict resolution* workload, by reducing the number of conflict points. It is easy to incorporate conflict resolution workload in the model, as we discuss below.

Conflict Resolution Workload We sub-divide the region (spanned by the sectors) into (reasonably) small size cells and compute the max-workload in each cell separately. If the size of the cell is small enough, a high max-workload cell would represent a conflict point, as more airplanes come in the vicinity of one another simultaneously. We add these cells as new (artificial) sectors to the data set and try to minimize their workload vector separately, thus (pos-

Grid Size	Set1 (Given SMax: 22)					Set2 (Given SMax: 18)				
	Given		Shifted			Given		Shifted		
	GMax	GMean	SMax	GMax	GMean	GMax	GMean	SMax	GMax	GMean
0.1×0.1	4	1.670	18	3	1.604	4	1.467	14	4	1.478
0.2×0.2	5	2.446	18	4	2.356	5	2.105	14	4	2.083

Table 12: Results of Right-Shift heuristic with additional grid constraints for Set1 and Set2. SMax: Sector Max, SMean: Sector Mean, GMax: Grid Max, GMean: Grid Mean.

sibly) decreasing the number of conflict points.

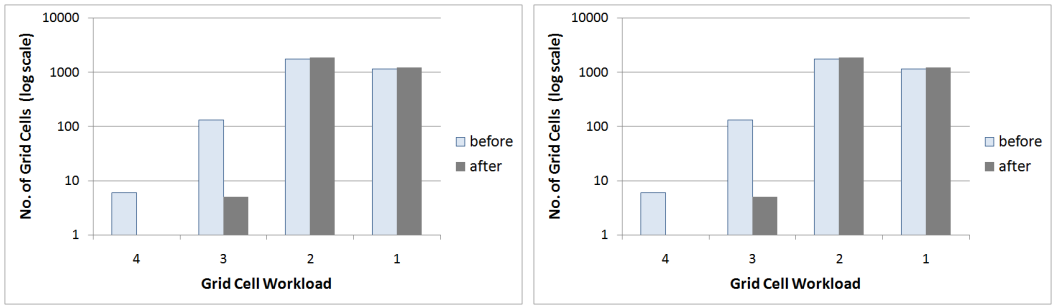
The shifting heuristic is now modified to be a two step procedure. First step treats the overall max value of max-workload across all cells as a constraint W_c. The air-crafts are re-scheduled to improve the workload vector of the sectors, like before, while keeping the workloads in all cells below W_c. In the second step, the role of sectors and cells is reversed. Now the optimized maximum value of the workload of the sectors is treated as a constraint and the air-crafts are re-scheduled with the objective of improving the workload vector of the cells.

For experimentation, these cells come from a uniform (square) grid and a shifted uniform grid as shown in Figure 44 covering the region spanned by the sectors. Two different side lengths of square grid cells are used, 0.1×0.1 and 0.2×0.2 (unit latitude/longitude degrees). In Set1, Set2 and Set5, 1 degree corresponds to somewhere in the range of $35 - 60$ nautical miles. Tables 12 and 13 shows the results of the workload improvements with the cell constraints. We observe that the max-workload of the sectors still improve as compared to the original (18 v/s 22 for Set1), while the number of conflict points are considerably decreased (see Figure 45). For Set1, after scheduling there are no grid cells with workload 4, while the number of cells with workload 3 has also decreased by more than 90%.

Concluding Remarks We presented a periodic flight plan scheduling problem, proved it to be NP-hard, and proposed heuristics for which we reported experimental results on real-world data. The results show a considerable workload improvement over the originally scheduled flight times and come at low computational cost. The reduction in the number of conflict points was also impressive. Future work will specifically aim to improve the lower bound, as

Grid Size	Set5 (Given SMax: 24)				
	Given		Shifted		
	GMax	GMean	SMax	GMax	GMean
0.1×0.1	11	1.609	19	8	1.598
0.2×0.2	14	2.271	19	10	2.243

Table 13: Results of Right-Shift heuristic with additional grid constraints for Set5. SMax: Sector Max, SMean: Sector Mean, GMax: Grid Max, GMean: Grid Mean.



(a) Set1. (b) Set2.

Figure 45: Grid cell max-workloads (before and after scheduling), for grid size 0.1×0.1.

we believe that the heuristically produced solutions are already almost optimal. Also, we are interested in re-routing the aircraft along with re-scheduling to improve the workloads.

4.2 Trajectory Clustering

Given the aircraft trajectories (historically flown, wind-optimized or any simulated data), it is important to identify the dominant flows. Recall, from Section 2.2, that a dominant flow is a path used by many air craft. Identifying dominant flows is critical for re-routing traffic in case of weather changes, designing sector boundaries (refer Section 2.3), realization of airspace concepts like tubes [52], etc. This is joint work with Irina Kostitsyna.

The main objective (similar to many other classical clustering problems) is to cluster trajectories into bundles and to find a representative for each cluster, defining the dominant flow. Note that the trajectory data may have many outliers (the trajectories that do not fly through or follow any dominant flow). Considering this fact, the decision version of the trajectory clustering problem can, now, be defined as:

Problem Statement: Given a set T of n aircraft trajectories (polygonal paths in $2D$), does there exist a set of dominant flows $D \subset T$, $|D| \leq k$, such that at least c % of trajectories in T lie "close" (within ϵ) to at least one of the dominant flows. A trajectory t is "close" to a dominant flow d if a certain fraction f, of its length $l(t)$, lies within the ϵ-fattening of d (see Figure 46). One may wish to minimize either k or minimize $\sum_{i \in D} l(i)$, where $l(i)$ is the length of dominant flow i. Alternately, given k the objective may be to maximize c.

Related Work [3] is an excellent web survey covering different kinds of trajectory clustering problems. *Repetition* is how Andrienko et al. [8] classify the trajectory clustering problem (as motivated above). Gudmundsson et al's book chapter [29] also gives a good overview of the topic, along with some key applications. Dykes et al. [26] suggest that the track density map approach discussed below dates back to the work of Hägerstrand in 1970.

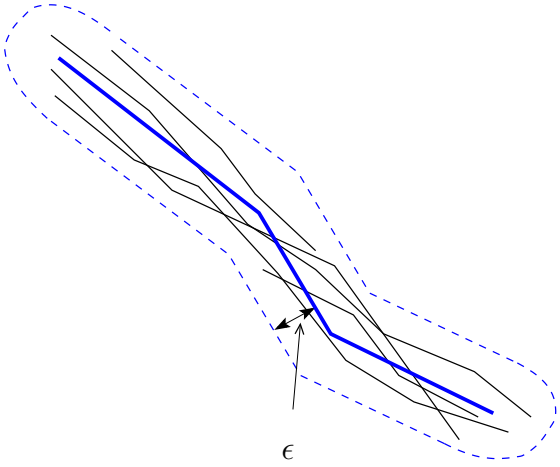


Figure 46: The (black) trajectories that are fully contained ($f = 1.0$) in ϵ-fattening of (blue) dominant flow, are "close" to the dominant flow.

Specific Requirements Related to Sector Design For sectorization purposes, along with the identification of flows, it is also important to assign speed and direction to each flow. This assists in deciding the buffer requirements for conflict points (flow crossings) from the sector boundary. One way of getting the speed (direction) of a dominant flow is by averaging (majority voting) the speed (direction) of tracks "close" to this flow.

4.2.1 Algorithms

We devise three greedy heuristics: GreedyD, GreedyW and GreedyT. The first two use the traffic density map, while the third one explicitly uses the given trajectories to find the dominant flows.

Traffic Density Map The construction of traffic density map is as follows. Starting with the bounding rectangle of trajectory set T, discretize the space of the rectangle with a uniform grid G. The spacing between points in G is defined by a parameter *incr*. For each grid point g define the weight $w(g)$ as the number of tracks that intersect a disc of radius ϵ (from problem definition) centered at this point. See Figure 50 for examples of traffic density maps. The points with heavy weight indicate regions of high traffic density.

Now, for each track t, we define the weight $w(t)$ as the sum of weights of all grid points that are within distance ϵ (in the ϵ-band) of t. The density of t is defined as $d(t) = w(t)/n(t)$ where $n(t)$ is the number of grid points in the ϵ-band of t. Intuitively, it is weight per unit length of the track and is a good quantifier for the dominance of t. A track t' with high $d(t')$, suggests that this track goes through (or lies completely in) the region of high traffic density. Thus, t' is a good candidate for dominant flow.

Algorithm 2 Greedy Trajectory Clustering

```
Input: Set T of tracks, Coverage threshold c %.
Output: Set D of dominant flows.
D ← φ
C ← φ
Coverage ← 0
while Coverage < c % do
  d ← "best" track
  D ← D ∪ d
  for all Tracks t ∈ T do
    if t ∉ C and t "close" to d then
      C ← t
    end if
  end for
  Coverage ← Σ_{t∈C} l(t) / Σ_{t∈T} l(t)
end while
```

All the three greedy heuristics are based on Algorithm 2, which is motivated from greedy set-cover. The only difference is the way in which each heuristic selects the "best" track. For GreedyD, the "best" track is one with the maximum density. Since a track with the maximum density may have a very low weight $w(t)$, it is not a good choice because this low weight track will hardly cover any tracks. We modify GreedyD by introducing a constraint that the weight of the selected track is at least a fraction wc of the maximum track weight. Thus, $d \leftarrow max_{\forall(t \in T \text{ and } t \notin C \text{ and } w(t) > wc \cdot w(t'))} d(t)$, where $t' \leftarrow max_{\forall(t \in T \text{ and } t \notin C)} w(t)$.

GreedyW selects the track with maximum weight, under the constraint that $d(t)$ is at least a fraction dc of the maximum track density. Thus, $d \leftarrow max_{\forall(t \in T \text{ and } t \notin C \text{ and } d(t) > dc \cdot d(t'))} w(t)$, where $t' \leftarrow max_{\forall(t \in T \text{ and } t \notin C)} d(t)$. Note

that the track density map is updated in each iteration of Algorithm 2, to have weights corresponding to tracks T/C. GreedyT, picks the track with maximum coverage, $d \leftarrow max_{\forall(t' \in T \text{ and } t' \notin C)} cov(t')$, where
$cov(t') = sum_{\forall(t \in T \text{ and } t \notin C \text{ and } t \text{ "close" to } t')} l(t)$.

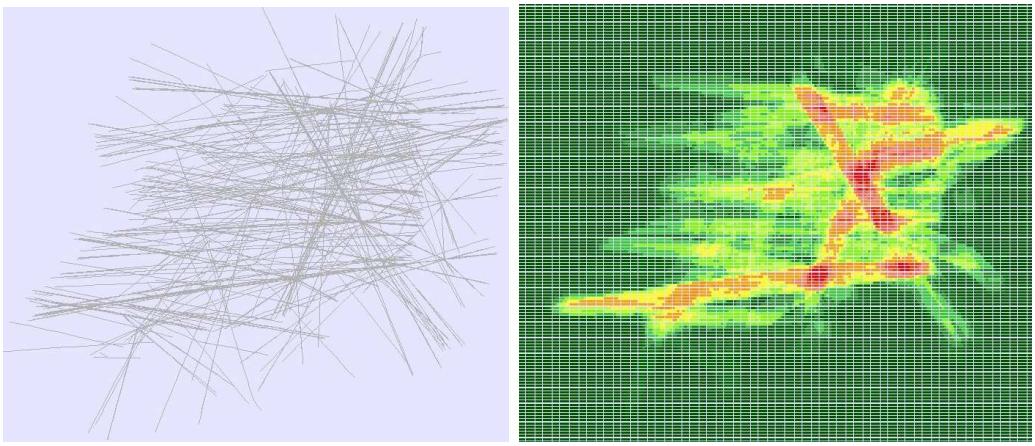
4.2.2 Experiments

Datasets **Set1**, **Set2** and **Set3** are same as used in Section 2.3.3. Refer to Figure 50, for the traffic density map of **Set1** and **Set2**, for $incr = 0.04$ and $\varepsilon = 0.15$. For all experiments below, $f = 0.7$. Thus, a track that was fractionally outside the ε-band of a dominant flow d, would still be covered by d.

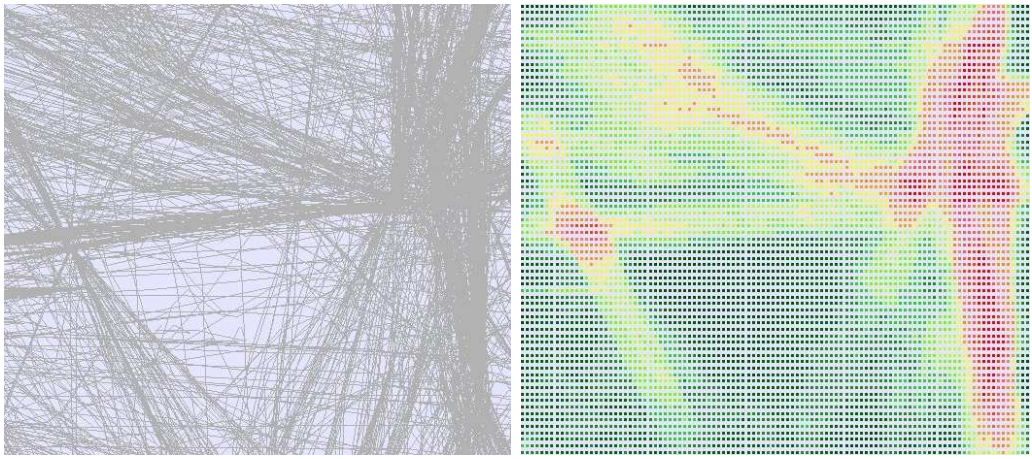
Firstly, we use **Set1** and **Set2** to evaluate the sensitivity of wc (and dc) with respect to k (number of dominant flows), for $c = 95\%$. Ideally, $incr$ should be as less as possible (more discrete points to approximate the continuous traffic density map) and ε should dictate the width of a dominant flow in nautical miles (nm). For this experiment, we use high values for $incr = 0.04$ and $\varepsilon = 0.15$ ($\sim 7.5nm$), as the main aim of experiment is to understand the sensitivity of parameters and not to extract the exact dominant flows. With high values of $incr$ and ε, one experiment for **Set1**, for a specific wc took less than 3 minutes to execute. In Figure 48, we show the variation of k as wc (and dc) increases from 0 to 1. The decreasing behavior of k, for GreedyW, is intuitive; as for lower values of wc, it is likely that the tracks with small $w(t)$ are selected initially, which do not cover many tracks. Similar explanation can be given for the increasing behavior of k (with decreasing wc), for GreedyD. Observe that, k remains same (75 for **Set1**) for both GreedyD, when $wc = 0$, and GreedyW, when $dc = 1$. This is because, both heuristics pick maximum density track (regardless of the weight), at each step. In a similar way, k remains same (44 for **Set1**) in case of GreedyD, when $wc = 1$, and GreedyW, when $dc = 0$, as maximum weight track is picked.

GreedyT, for $c = 95\%$, gave 28 and 35 dominant flows for **Set1** and **Set2**, respectively. Comparing GreedyT results with the graphs in Figure 48, we assign $dc = 0.6$ and $wc = 0.7$. These are chosen so that the number of dominant flows k, for GreedyD and GreedyW, remains comparable to that achieved by GreedyT; at the same time, the density of each dominant flow is as high as possible.

Next, we use the constraint parameters fixed above, to see how k changes

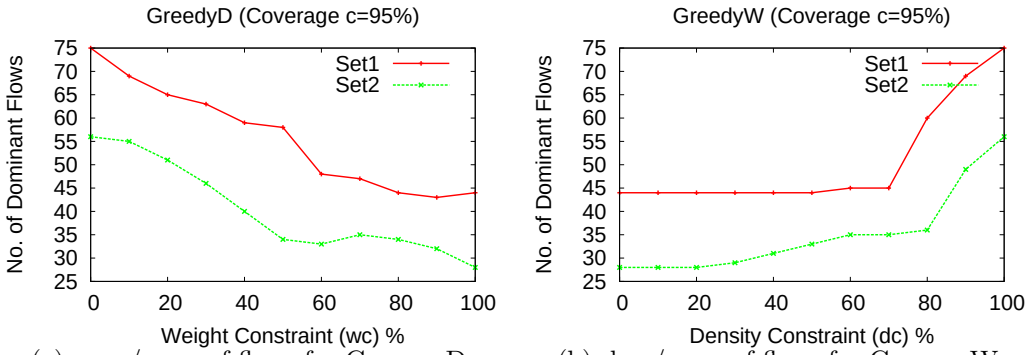


(a) **Set1**.



(b) **Set2**.

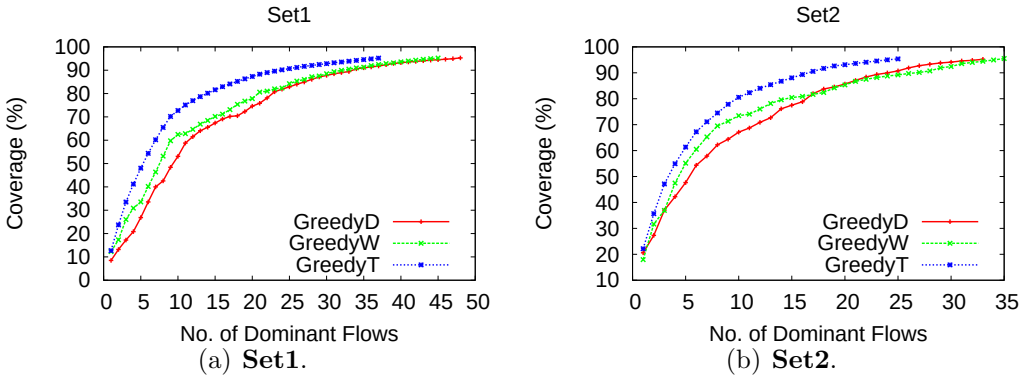
Figure 47: Left: Tracks; Right: Traffic Density Map showing regions with high (red), moderate (yellow) and very low (dark green) traffic density.



(a) wc v/s no. of flows for GREEDYD. (b) dc v/s no. of flows for GREEDYW.

Figure 48: Sensitivity of number of dominant flows to the constraints.

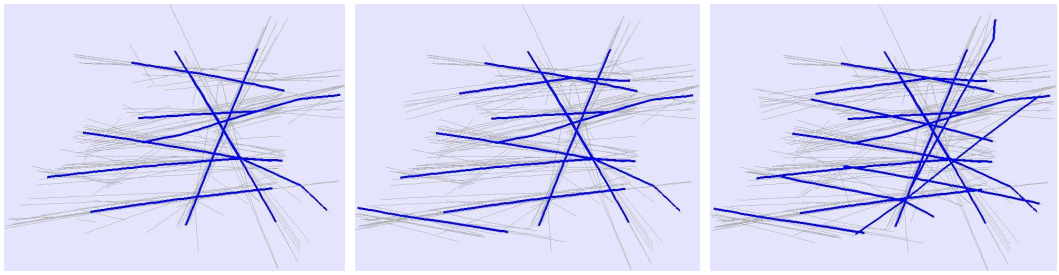
with the increasing coverage (c). Refer Figure 49: For $c \geq 60\%$, the addition of a flow, does not increase the coverage by much. Other way to interpret this is, for coverage beyond 60%, the density (dominance) of any new flow is modest. Visual inspection (see Figure 49) of dominant flows at different coverage levels also reveals that 60% is the right choice for c.



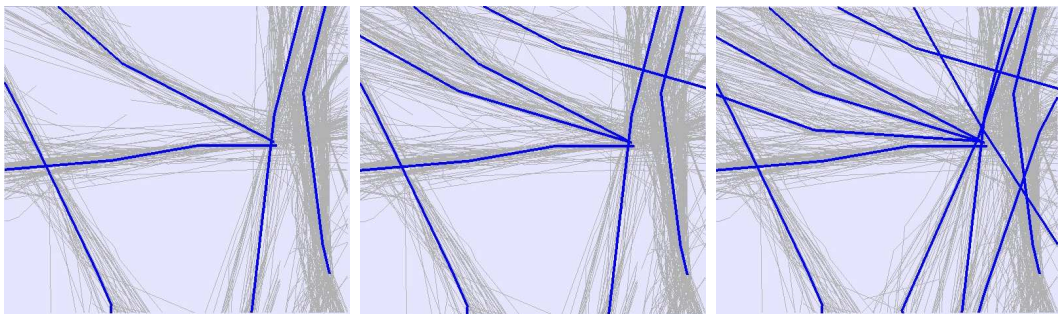
(a) **Set1**. (b) **Set2**.

Figure 49: Number of dominant flows vs coverage.

Refer to Figure 51, for the results of GREEDYD and GREEDYW, for $c = 60\%$, for the **Set3**. For this experiment, $incr = 0.015$ and $\varepsilon = 0.05$ ($\sim 2.5nm$). Most of the high traffic density regions look covered with the computed domi-



(a) **Set1**.



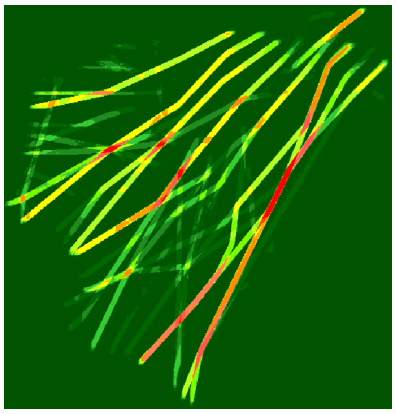
(b) **Set2**.

Figure 50: Screenshot of (blue) dominant flows (along with covered (grey) tracks) for increasing total coverage (Left: $c = 50\%$, Middle: $c = 60\%$ and Right: $c = 70\%$).

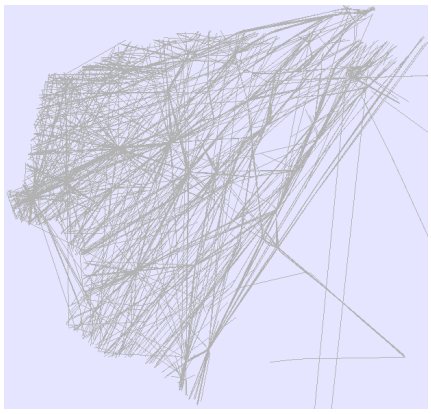
nant flows.

Concluding Remarks The requirement, $D \subset T$, in the problem statement might be too rigid. It will be interesting to identify a dominant flow which is central to the tracks that lie within its ϵ-band. This is similar to considering the centroid of points as cluster head (like done in k-means clustering). More formally, the sub-problem can be thought of as: Given a set T', of polygonal chains in $2D$, find a chain t so as to $\min \sum_{i \in T'} d(t, i)$ or $\min \max_{i \in T'} d(t, i)$, where $d(t, i)$ is the Fréchet distance between t and i. A possible related reference is the work of Bereg et al [12], where they consider finding Voronoi diagram of polygonal chains under discrete Fréchet distance.

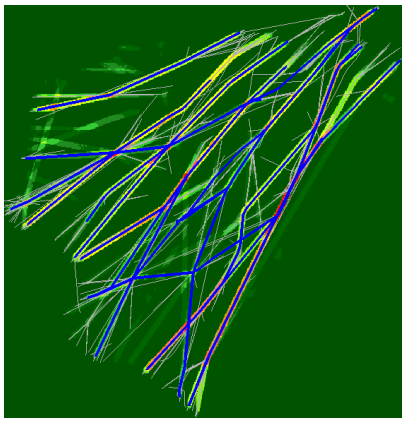
Also, for sectorization purposes, it may suffice to get the output in the form of a planar graph (defining the arrangement of dominant flows), with edges (poly-chains) connecting nodes (flow crossing points). The direction and speed of tracks along the edges can be used to specify the separation requirement of crossing points from sector boundaries. Such output, in terms of planar graph, can be expected from density based clustering methods [48]. Experimentation with such methods is left as a future work.

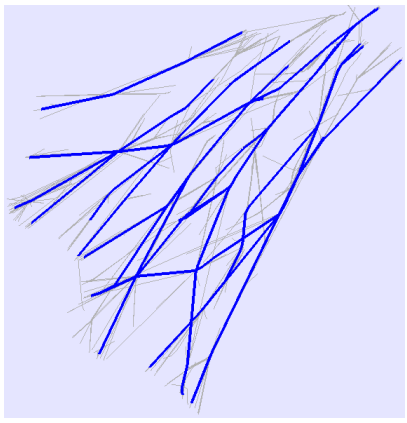


(a) Traffic density map.

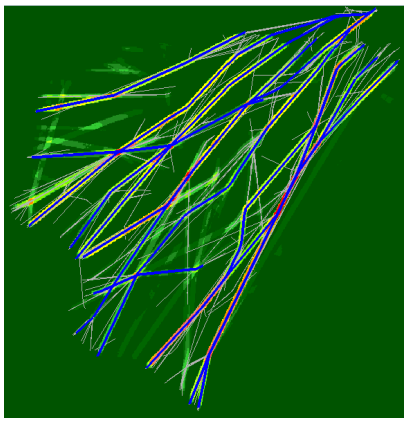


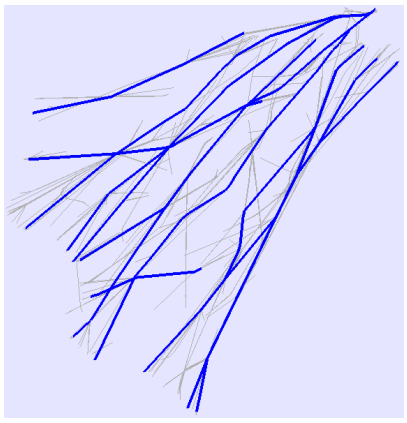
(b) Track data.





(c) Dominant flows (blue) produced by GreedyD (Left: Overlayed on density map; Right: Along with the covered trajectories).





(d) Dominant flows (blue) produced by GreedyW (Left: Overlayed on density map; Right: Along with the covered trajectories).

Figure 51: Screenshots of result for **Set3** ($c = 60\%$).

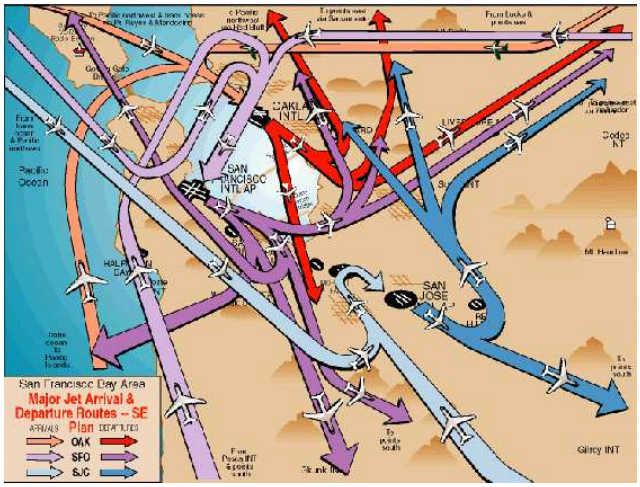
5 Future Work

Terminal Airspace All of the methods discussed so far (and the experiments conducted) were aimed at sectorizations involving en route airspace. The terminal airspace (within $30 - 50$ nautical miles of an airport) is sectorized very differently from the en route airspace. Each Terminal Control Areas (TCA) (sector in the terminal airspace) is very structured to enable efficient climb and descent traffic profiles.

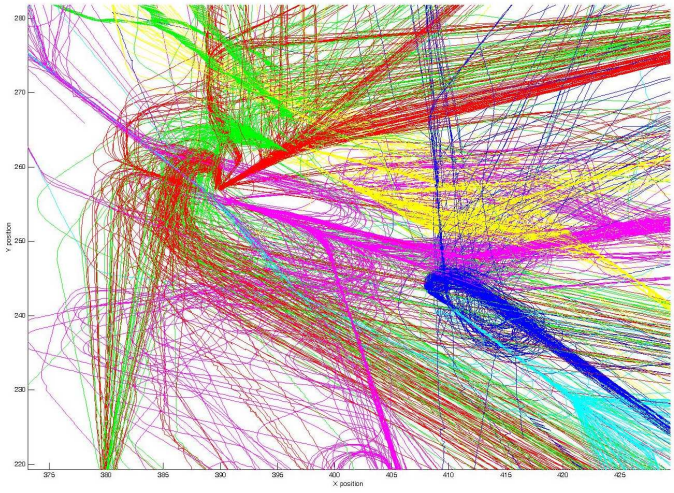
As discussed in Bert et. al [30], runway and flow path design is inseparably intertwined with sectorization in the terminal airspace. Refer to Figure 52 for an example of highly structured flow interactions in the SFO/OAK/SJC metroplex area. (The images are from the terminal airspace slides of Doug Isaacson, NASA Ames Research Center.) The function of controllers in TCA's is very specific to the kind of flow they are handling. There are feeder sectors that handle TRACON entry and merge preparation, final approach sectors, departure sectors, hand-off sectors, etc.

The study of combined flow design and sectorization is new and challenging. We are currently compiling exact requirements and exploring algorithmic approaches for terminal airspace design.

Airspace Playbook Kopardekar et. al [37] proposed the concept of an "Airspace Playbook" as a set of pre-defined airspace configurations that go hand-in-hand with the the existing pre-defined scenarios from the National Severe Weather Playbook (NSWP). The NSWP consists of a set of reroutes based on weather changes. Each "configuration play" involves adjustment to sector boundaries, ideally in the increments of existing sector components (sub-sectors). Typical examples of a play include exchanging sub-sectors among neighboring sectors, splitting sub-sectors within a sector etc. The main advantage of pre-defined airspace configuration lies with the "memorization" [18] aspect of controller training. Since sub-sectors are used in configuration plays, air traffic controllers can train themselves on sub-sector geometry.



(a) Planned flows.



(b) Actual flight tracks.

Figure 52: Terminal flow interaction in SFO/OAK/SJC.

The local repartitioning method, described in Section 3.3, restricts the redesign to regions of dynamic air traffic; each redesign consists of a local move, such as $(2 \rightarrow 2)$, $(2 \rightarrow 3)$, $(2 \rightarrow 1)$, etc. While the set of local moves is pre-defined (and small), each local move may result in many different configurations in the same region. This is because of the large number of possible flow conforming cuts available in the discrete search space (uniform grid). That said, it is not difficult to incorporate the concept of sub-sectors into the local repartitioning method used by GeoSect-D. Instead of searching for a new cut on a discrete grid, the search graph may consist of nodes and edges that define sub-sector boundaries within a sector (more generally, a region). This will restrict the number of different possible configurations (resulting from a local move) to a small set of different combinations of sub-sectors. A challenging question is to come up with a good set of candidate sub-sectors that define an optimal underlying search graph for cuts. One way to get the sub-sectors is to invoke GeoSect for designing a large number of sectors. Another, more flow conforming, way would be to use the Voronoi partition (see Figure 53) of dominant flows as the set of candidate sub-sectors.

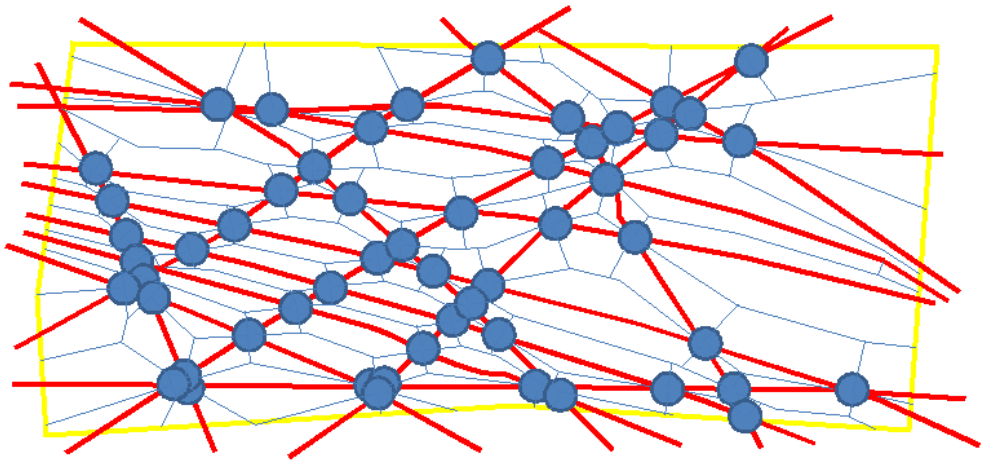


Figure 53: (Grey blue) Voronoi partition of (red) dominant flows. The (blue) discs represent constraint zones at flow crossing points.

Multi-Controller Staffing Tien et al [54] discuss the use of a multi-controller policy where, by assigning more than one controller to a sector, they circumvent the need to perform disruptive sector boundary changes during busy periods of air-traffic. The addition of an extra controller may not double the

capacity of a sector, but can definitely serve as an alternative to re-design, when the workload of sector increases above that manageable by one controller. This adds another interesting dimension to the airspace sectorization problem, where the new objective is to minimize the controller-hours (which directly translates to the cost of controller-staffing). The current methods in GeoSect can be appropriately modified to handle this new objective; among all choices of flow conforming cuts, instead of picking the one that best balances the workload, we may prefer a cut that gives a better controller staffing opportunity.

GeoSect **Enhancements** There are several specific planned enhancements in the software, including:

- **GUI for manually editing the sector boundaries:**
  Add the capability for a user to edit (delete/add/move) a sector boundary. While a boundary is being edited, the workload of the sectors sharing that boundary should be updated in real time, allowing instant assessment of possible benefits. This feature is important for multiple reasons as enumerated below.

  1. Post-process a computer generated sector design, either from GeoSect or from any other algorithm. This should assist in rectifying any small artifacts in the design.
  2. Validate the methods that generated the sector design. If by editing a sector boundary, the workload balance improves, one may want to re-examine or modify the algorithm, adjust the algorithm's parameters, or check for the correct implementation of the algorithm. Alternately, if the method is working correctly, this might be another way for a user to understand why the algorithm arrived at a particular sector design.
  3. Allow the possibility for the user to decide to skip considering an auto-generated sector design altogether. This may be particularly useful in the dynamic setting, where a sector design is already in use. Modifying the sector boundary manually, while monitoring the changes in the resulting workload, may well end up being the method of choice, especially for small changes in traffic patterns.

  The important data structure, *doubly connected edge list (DCEL)*, to make possible these editing features is already present in GeoSect. Hence, it should be one of the easier enhancements to the tool.

- **Using dominant flows more intelligently:**
  Update the current implementation of the constraint at flow crossing points. Currently, the constraint is implemented as a circular disc. The modification will allow other shaped regions (e.g., rectangles, polygons) that will allow consideration of the direction and speed of the crossing flows (see Section 2.3). We also intend to make necessary modifications to consider the dominant flows in $3D$ (and possibly $4D$). This will be important for producing sector designs in the TRACON region and also for dynamic re-sectorization.

- **Starting with an input sector design:**
  The local heuristics, mentioned in Section 3.3.1, allow GeoSect to start with an input sector design and adjust the sector boundaries locally, to either improve the workload or to make them flow conforming. A globally optimal solution (e.g., MIP [59]) that gives good workload balance, but has issues with sector boundary flow interactions, is likely to benefit from such post-processing.

  More specifically, in order to make use of a global solution, it will be important for GeoSect not only to make the boundary changes that are absolutely necessary, but also to keep the resulting new design as close as possible to the input design. To achieve this goal, the local 2-opt method can be modified as follows. After erasing a boundary, increase the weight of (or add more) search nodes near this boundary. By increasing the weight, we mean that a node near this boundary should be more likely to be on the new cut. Any constraint violation by the new cut should be addressed, regardless to this weighing scheme.

- **Extending local re-partitioning method:**
  Add more options for a local move, including $(2 \rightarrow 1)$, $(2 \rightarrow 3)$ $(1 \rightarrow 2)$ etc.

- **Running time:**
  Running time was not an issue with GeoSect1.0, which produced convex sector designs very quickly in practice. However, incorporating flow (and other) constraints in the model has resulted in a substantial increase in the running time. For dynamic re-sectorization, it is critical to generate the sector designs quickly. Real-time update of workload (and constraint violation checks) is also inevitable for adding the manual editing feature. While running time was not our primary focus in recent

developments designed to improve functionality and quality of sectors, there are many optimizations we expect to incorporate in the software to increase its efficiency.

References

[1] National airspace redesign (NAR). *Office of Air Traffic and Airspace Management, Federal Aviation Administration*.

[2] Occupational outlook handbook. *Bureau of Labor Statistics, U.S. Department of Labor*.

[3] `http://movementpatterns.pbworks.com/Patterns-of-Movement`.

[4] `http://voronoi.ams.sunysb.edu/~gk/projects/GeoSect`.

[5] Human operator or load on air traffic control. *M. W. Somlensky and E. S. Stein, editors, Human factors in air traffic control*, pages 155–183, March 1998.

[6] M. Altman. Is automation the answer? The computational complexity of automated redistricting. *Rutgers Computer and Law Technology Journal*, 23(1):81–142, 1997.

[7] M. Altman and M. McDonald. A computation-intensive method for evaluating intent in redistricting. In *2004 Midwest Political Science Association Conference*, Chicago, IL, April 2004.

[8] N. Andrienko and G. Andrienko. Designing visual analytics methods for massive collections of movement data. In *Cartographica v.42 (2)*, pages 117–138, 2007.

[9] B. Aronov, P. Carmi, and M. J. Katz. Minimum-cost load-balancing partitions. In *SCG '06: Proceedings of the 22nd annual Symposium on Computational Geometry*, pages 301–308, New York, NY, USA, 2006. ACM.

[10] N. Bansal, M. Mahdian, and M. Sviridenko. Minimizing makespan in no-wait job shops. *Mathematics of Operations Research*, 30(4):817–831, 2005.

[11] A. Basu, J. S. B. Mitchell, and G. Sabhnani. Geometric algorithms for optimal airspace design and air traffic controller workload balancing. In *Ninth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 75–89, San Fransisco, CA, January 2008.

[12] S. Bereg, K. Buchin, M. Buchin, M. L. Gavrilova, and B. Zhu. Voronoi diagram of polygonal chains under the discrete fréchet distance. In *Proceedings of the 14th annual International Conference on Computing and Combinatorics*, pages 352–362, 2008.

[13] P. K. Bergey, C. T. Ragsdale, and M. Hoskote. A simulated annealing genetic algorithm for the electrical power districting problem. *Annals of Operations Research*, 121(1–2):33–55, July 2003.

[14] P. Berman, B. Dasgupta, and S. Muthukrishnan. On the exact size of the binary space partitioning of sets of isothetic rectangles with applications. *SIAM Journal of Discrete Mathematics*, 15:252–267, 2002.

[15] P. Berman, B. Dasgupta, S. Muthukrishnan, and S. Ramaswami. Improved approximation algorithms for rectangle tiling and packing. In *Proceedings of the 12th annual ACM-SIAM Symposium on Discrete Algorithms*, pages 427–436, 2001.

[16] D. Bertsimas, G. Lulli, and A. Odoni. The air traffic flow management problem: An integer optimization approach. In *13th International Conference on Integer Programming and Combinatorial Optimization, IPCO 2008 Bertinoro*, volume 5035, pages 34–46, May 2008.

[17] M. Bloem and P. Kopardekar. Combining airspace sectors for the efficient use of air traffic control resources. In *AIAA Guidance, Navigation, and Control Conference*, Aug 2008.

[18] C. R. Brinton and L. S. Cook. Analysis of current airspace operations and implications for dynamic airspace configuration. In *AIAA Guidance, Navigation, and Control Conference*, Aug 2008.

[19] T. M. Chan. Geometric applications of a randomized optimization technique. *Discrete & Computational Geometry, ACM*, 22(4):269–278, December 1999.

[20] T. M. Chan. Personal communication, 2006.

[21] R. Cole, J. S. Salowe, W. L. Steiger, and E. Szemerédi. An optimal-time algorithm for slope selection. *SIAM Journal on Computing*, 18(4):792–810, 1989.

[22] R. S. Conker, D. A. Moch-Mooney, W. P. Niedringhaus, and B. T. Simmons. New Process for "Clean Sheet" Airspace Design and Evaluation. In *7th US/Europe ATM Seminar*, July 2007.

[23] A. D'Ariano, D. Pacciarelli, and M. Pranzo. A branch and bound algorithm for scheduling trains in a railway network. In *European Journal of Operational Research*, volume 183(2), pages 643–657, December 2007.

[24] B. Dasgupta and S. Muthukrishnan. Slice and dice: A simple, improved approximate tiling recipe. In *Proceedings of the 13th annual ACM-SIAM Symposium on Discrete Algorithms*, pages 455–464, 2002.

[25] W. F. de la Vega and G. Lueker. Bin packing can be solved within $1 + \epsilon$ in linear time. *Combinatorica*, 1(4):349–355, 1981.

[26] J. A. Dykes and D. M. Mountain. Seeking structure in records of spatio-temporal behaviour: visualization issues, efforts and applications. *Computational Statistics & Data Analysis*, 43(4):581–603, 2003.

[27] A. H. Farrahi and Z. Wood. Computational complexity of the airspace sectorization problem. Personal Communication, Feb 2009.

[28] S. L. Forman and Y. Yue. Congressional districting using a TSP-based genetic algorithm. *Lecture Notes in Computer Science*, 2724:2072–2083, Jan 2003.

[29] J. Gudmundsson, P. Laube, and T. Wolle. Movement patterns in spatio-temporal data. In *Encyclopedia of GIS*, pages 726–732. 2008.

[30] B. Hackney, R. Hoffman, B. Khorrami, R. Kicinger, T. Lewis, M. Lowther, J. Mitchell, J. Prete, G. Sabhnani, K. Stefanidis, and A. Yousefi. Airspace Algorithms: Dynamic Airspace Configuration DCN 32N0209-014. Technical report, NASA Ames Research Center, March 2009.

[31] S. Har-Peled. Clustering motion. *Discrete & Computational Geometry*, 31(4):545–565, 2004.

[32] K. C. Hendy, J. Liao, and P. Milgram. Combining time and intensity effects in assessing operator information and processing load. *Journal of Human Factors*, 39(1):30–47(18), 1997.

[33] D. Karger, C. Stein, and J. Wein. Scheduling algorithms. *CRC Handbook of Computer Science*, 1997.

[34] S. Khanna, S. Muthukrishnan, and M. Paterson. On approximating rectangle tiling and packing. In *Proceedings of the 9th annual ACM-SIAM Symposium on Discrete Algorithms*, pages 384–393, Philadelphia, PA, USA, 1998. Society for Industrial and Applied Mathematics.

[35] S. Khanna, S. Muthukrishnan, and S. Skiena. Efficient array partitioning. In *ICALP '97: Proceedings of the 24th International Colloquium on Automata, Languages and Programming*, pages 616–626, London, UK, 1997. Springer-Verlag.

[36] J. Kim, A. Kröller, J. S. B. Mitchell, and G. R. Sabhnani. Scheduling aircrafts to reduce controller workload. *Submitted to 17th annual European Symposium on Algorithms (ESA)*, 2009.

[37] A. Klein, P. Kopardekar, M. D. Rodgers, and H. Kaing. "Airspace Playbook": Dynamic Airspace Reallocation Coordinated with the National Severe Weather Playbook. In *AIAA 7th Aviation Technology, Integration and Operations (ATIO) Forum*, Belfast, UK, September 2007.

[38] S. Klein, M. Rodgers, H. Kaing, P. Lucic, and K. Leiden. DAG CE-6 Golden Nuggets Fast-Time Modeling and Simulation Studies Final Report Part 3: Dynamic Airspace Configuration Analysis. Technical report, NASA Ames Research Center, December 2008.

[39] K. Lee. *Describing Airspace Complexity: Airspace Response to Disturbances*. PhD thesis, Georgia Institute of Technology, April 2008.

[40] P. M. Lennartz. *No-Wait Job Shop Scheduling, a Constraint Propagation Approach*. PhD thesis, UU Universiteit Utrecht, Netherlands, 2006.

[41] A. Mascis and D. Pacciarelli. Job-shop scheduling with blocking and no-wait constraints. *European Journal of Operational Research*, 143(3):498–517, December 2002.

[42] J. S. B. Mitchell, G. Sabhnani, J. Krozel, B. Hoffman, and A. Yousefi. Dynamic airspace configuration management based on computational geometry techniques. In *AIAA Guidance, Navigation, and Control Conference*, Aug 2008.

[43] R. H. Mogford, J. A. Guttman, S. L. Morrow, and P. Kopardekar. The complexity construct in air traffic control: A review and synthesis of the literature. Technical report DOT/FAA/CT-TN95/22, Department of Transportation, Federal Aviation Administration Technical Center, July 1995.

[44] S. Muthukrishnan, V. Poosala, and T. Suel. On rectangular partitionings in two dimensions: Algorithms, complexity, and applications. In *ICDT '99: Proceedings of the 7th International Conference on Database Theory*, pages 236–256, London, UK, 1999. Springer-Verlag.

[45] S. Muthukrishnan and T. Suel. Approximation algorithms for array partitioning problems. *Journal of Algorithms*, 54(1):85–104, 2005.

[46] M. S. Paterson and F. F. Yao. Efficient binary space partitions for hidden-surface removal and solid modeling. *Discrete & Computational Geometry*, 5(5):485–503, 1990.

[47] M. S. Paterson and F. F. Yao. Optimal binary space partitions for orthogonal objects. *Journal of Algorithms*, 13(1):99–113, 1992.

[48] J. Sander, M. Ester, H.-P. Kriegel, and X. Xu. Density-based clustering in spatial databases: The algorithm gdbscan and its applications. *Data Mining and Knowledge Discovery*, 2(2):169–194, 1998.

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

[50] C. J. Schuster. No-wait job shop scheduling: Tabu search and complexity of subproblems. *Mathematical Methods of Operations Research*, 63(3):473–491, July 2006.

[51] C. J. Schuster and J. M. Framinan. Approximative procedures for no-wait job shop scheduling. *Operations Research Letters*, 31(4):308–318, 2003.

[52] B. Sridhar, S. Grabbe, K. Sheth, and K. D. Bilimoria. Initial study of tube networks for flexible airspace utilization. In *AIAA Guidance Navigation and Control Conference*, Aug. 2006.

[53] D. Sweet, V. Manikonda, J. Aronson, K. Roth, and M. Blake. Fast-time simulation system for analysis of advanced air transportation concepts. In *AIAA Modeling and Simulation Technologies Conference and Exhibit*, Monterey, CA, August 2002.

[54] S.-L. A. Tien and R. Hoffman. Optimizing airspace sectors for varying demand patterns using multi-controller staffing. In *Eighth USA/Europe Air Traffic Management Research and Development Seminar*, June 2009.

[55] G. J. Woeginger. Inapproximability results for no-wait job shop scheduling. *Operations Research Letters*, 32(4):320–325, 2004.

[56] G. L. Wong. Analysis of different cost functions in the geosect airspace partitioning tool. *To be presented at the 28th Digital Avionics Systems Conference (DASC)*, October 2009.

[57] I. Wyndemere. Dynamic resectorization: Accommodating increased flight flexibility. Technical report, Boulder, CO, 1997.

[58] M. Xue. Airspace sector redesign based on voronoi diagrams. In *AIAA Guidance, Navigation, and Control Conference*, Aug 2008.

[59] A. Yousefi. *Optimum Airspace Design with Air Traffic Controller Workload-Based Partitioning*. PhD thesis, George Mason University, 2005.

[60] A. Yousefi and G. L. Donohue. Temporal and spatial distribution of airspace complexity for air traffic controller workload-based sectorization. In *AIAA 4th Aviation Technology, Integration and Operations (ATIO) Forum*, Chicago, IL, September 2004.