Geometric Algorithms for Optimal Airspace Design and Air Traffic Controller Workload Balancing

AMITABH BASU
Carnegie Mellon University
JOSEPH S. B. MITCHELL
Stony Brook University
and
GIRISHKUMAR SABHNANI
Stony Brook University

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. The novelty of the problem is that it partitions data consisting of trajectories of *moving* points, rather than static point set partitioning that is commonly studied. First, we formulate and solve the 1D version of the problem, showing how to partition a line into "sectors" (intervals) according to historical trajectory data. Then, we apply the 1D solution framework to design a 2D sectorization heuristic based on binary space partitions. We also devise partitions based on balanced "pie partitions" of a convex polygon.

We evaluate our 2D algorithms experimentally, applying our algorithms to actual historical flight track data for the NAS. 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. An even better improvement is seen in the standard deviations (over all sectors) of both time-average and worst-case workloads.

Categories and Subject Descriptors: F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Geometrical problems and computations*

General Terms: Algorithms

Authors' addresses: A. Basu, Tepper School of Business, Carnegie Mellon University, Pittsburgh, PA, 15213. J. Mitchell, Applied Mathematics and Statistics, Stony Brook University, Stony Brook, NY 11794-3600. G. Sabhnani, Computer Science, Stony Brook University, Stony Brook, NY 11794-4400.

J. Mitchell has been partially supported by grants from the National Science Foundation (CCR-0098172, ACI-0328930, CCF-0431030, CCF-0528209, CCF-0729019), the U.S.-Israel Binational Science Foundation (grant No. 2000160), Metron Aviation (NASA subcontract, NAS2-02075), and NASA Ames (NAG2-1620).

Additional Key Words and Phrases: Geometric partitioning, airspace sectorization, workload balancing, binary space partitions

1. INTRODUCTION

The National Airspace System (NAS) 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), each of which is partitioned into *sectors*, each one of which is managed by one air traffic controller (or a small team of 1-3 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 [Occupational Outlook Handbook], 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 [National Airspace Redesign (NAR)] 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; they are not the result of analysis of route structures and demand profiles, which have changed over the years, while the sector geometry has stayed relatively constant.

In this paper 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.

More formally, *the 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.

The quantification of "workload" is very important to the problem. Workload

is challenging to model accurately, since it should take into account human factors issues, which include subjective estimations of psychological/physiological state and mental effort. Mental workload involves issues of visual and auditory perception, memory, stress, and attention span. Many prior studies (see, e.g., [Hendy et al. 1997; Mogford et al. 1995; Stein 1998; Schmidt 1976]) 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 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 these other factors and add them to our model (see our discussion in Section 6).

The historical 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 [Sweet et al. 2002]), 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.

1.1 Related Work

The sectorization problem has been studied most recently as a global optimization problem using techniques of integer programming; after discretizing the NAS into 2566 hexagonal cells, Yousefi and Donohue [Yousefi and Donohue 2004; Yousefi 2005] formulate and solve (using CPLEX) a mixed integer 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. Delahaye et al. [Delahaye et al. 1998] use genetic algorithms for sectorization. Tran et al. [Tran et al. 2003] apply graph partitioning methods to sectorization.

In the algorithms literature, there has been related work on partitioning of rectangles and arrays for load balancing of processors; see, e.g., [Berman et al. 2000; 2002; Berman et al. 2001; Khanna et al. 1998; Khanna et al. 1997; Muthukrishnan et al. 1999; Muthukrishnan and Suel 2005].

Geographical load balancing applications have arisen in political districting (to avoid gerrymandering); see Altman [Altman 1997] (who proves NP-hardness of political districting), Altman and McDonald [Altman and McDonald 2004], and Forman and Yue [Forman and Yue 2003]. Geographic load balancing also arises in electric power districting; see, e.g., Bergey, Ragsdale, and Hoskote [Bergey et al. 2003]. Recent work in the computational geometry literature looks at minimum-

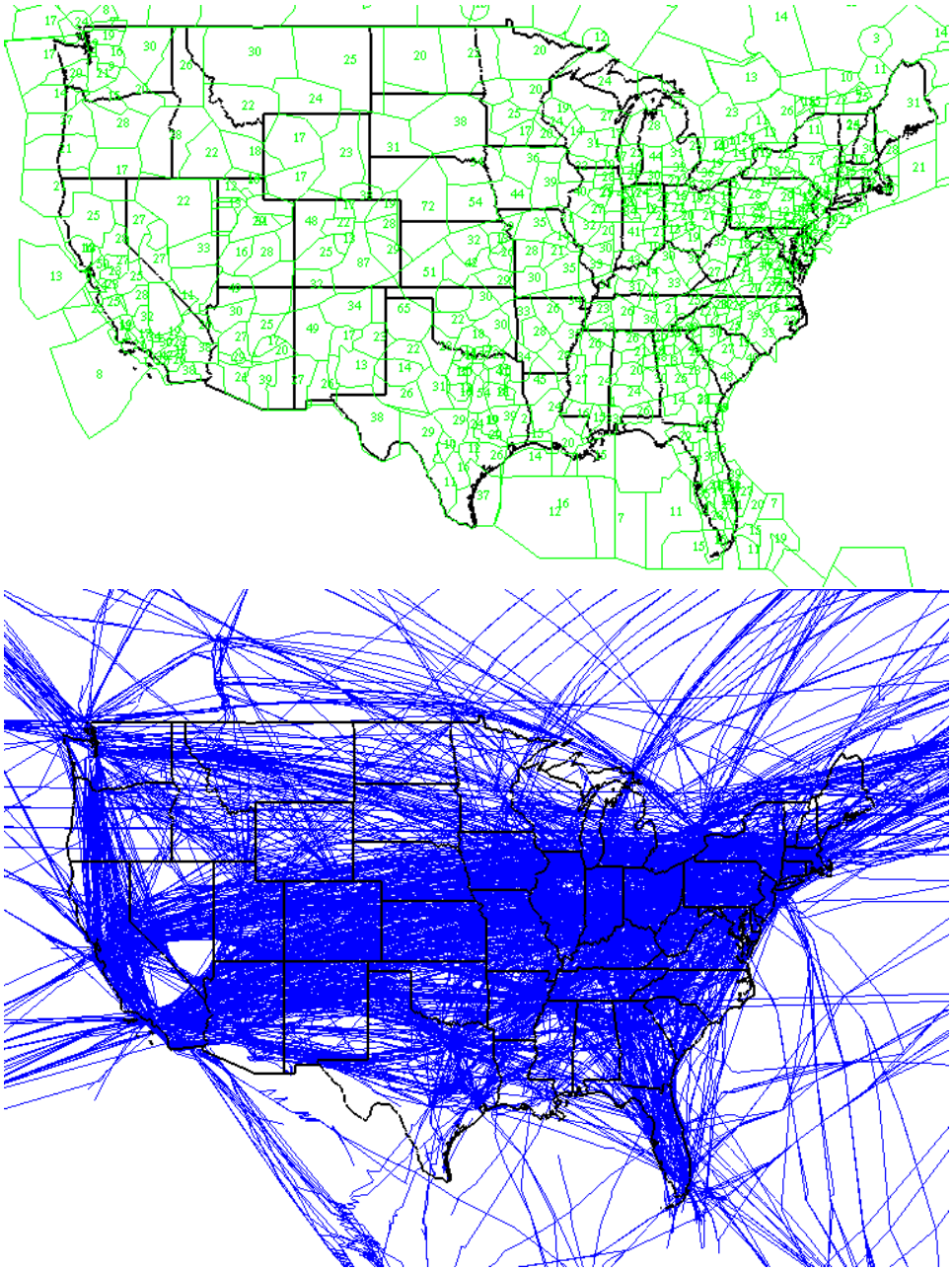

Fig. 1. Top: The current sectorization of the airspace over the USA. Bottom: Historical track data for flights.

cost load balancing in sensor networks; see Carmi and Katz [Carmi and Katz 2005].

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. This implies that a 2D version of our problem is really best thought of in 3D (x, y, t), and it means that even the 1D version of our problem has interesting structure, as it maps to a 2D partitioning problem in space-time.

1.2 Summary of Contributions

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

(2). We provide an exact solution to some versions of the one-dimensional (1D) sectorization problem.

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

(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 also compare the results we obtain with the existing sectorization currently in use by the FAA.

Our 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.

We have made various simplifications in developing the model and implementing our solutions. We are able to extend our solutions in several directions; see Section 6. Currently, our software works in only 2D; however, the methods extend to account for different altitudes and climb and descend trajectories. We also deal with only one altitude level of sectorization, the so-called "high altitude" sectors; there are also low altitude sectors for general aviation aircraft and ultra-high altitude sectors for military and certain transcontinental flights. In the terminal area near an airport, there are also more complex three-dimensional sectors corresponding to arrival and departure flights, which necessarily change altitude. We model workload in terms of aircraft density (number of aircraft in a sector), determined by a given set of track data, which may come from historical data (as ours did) or from the results of a simulation or from wind-optimized computed trajectories. In the conclusion (Section 6), we discuss extensions of our methods to include more realistic models of workload (e.g., that include *coordination* workload). We acknowledge that the results reported in this paper are based on a highly simplified workload estimation model; our results are, however, applicable to a broader model, and understanding how our methods behave in the simplified model may help in understanding the bigger picture, including the operational constraints that impact the sectorization problem.

2. THE 1D SECTORIZATION PROBLEM

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.

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 [Chan 1999]. 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 [Chan 2006]. 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. [Cole et al. 1989]), 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. □

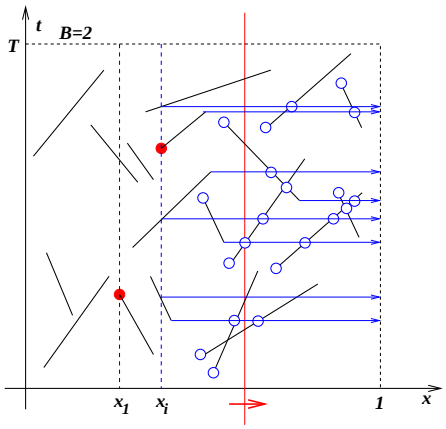

Fig. 2. Sweeping ℓ (red) rightwards. The hollow (blue) circles indicate critical points where the max-workload might increase as ℓ sweeps.

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.)

3. THE SECTORIZATION PROBLEM IN TWO DIMENSIONS

In contrast with prior work on partitioning sets of (static) points in the plane, or elements of an array (e.g., see [Khanna et al. 1998; Khanna et al. 1997; Muthukrishnan et al. 1999; Muthukrishnan and Suel 2005]), 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.

3.1 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 [Khanna et al. 1998], 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.

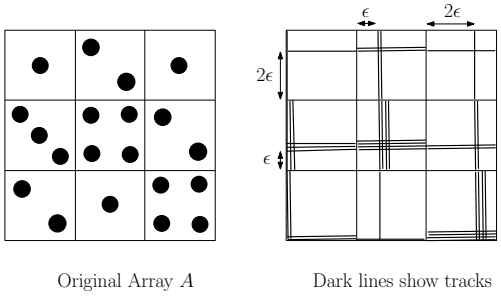

Fig. 3. Reduction from array partitioning to rectangular partitioning for a 3×3 array.

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

3.2 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 [Paterson and Yao 1990; 1992]. 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 3.2. *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 $w(o, i) = \text{Partition}(P_1) + \text{Partition}(P_2)$.

(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 $B(o, i, k_1, k_2) = \max\{\text{Partition}(P_1,k_1), \text{Partition}(P_2,k_2)\}$.

(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 [Berman et al. 2000] 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.

3.3 BSP Heuristic

Rather than implement 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 5.2.

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.

3.4 Avoiding Bad Aspect Ratios

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 may be in the sector for radically

different time periods, depending on the orientation of the trajectory with the diameter of the sector.

To address this issue, we use the aspect ratio of the sector 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. Refer to Figure 4 (top). However, this heuristic can still result in bad aspect ratios as shown in the Figure 4 (bottom).

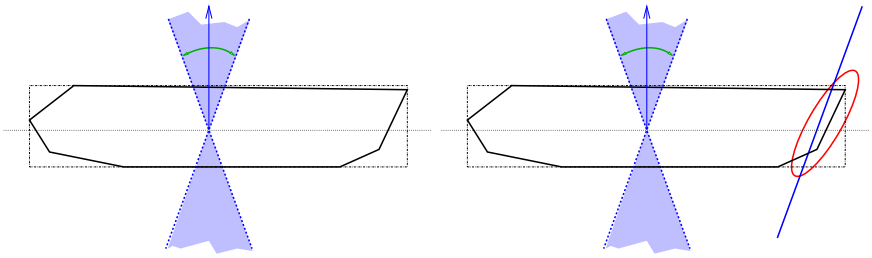

Fig. 4. Left: The allowable range for cut orientations is shaded. Right: A cut within the allowable orientation range may result in a skinny polygon on the right.

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.

3.5 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.

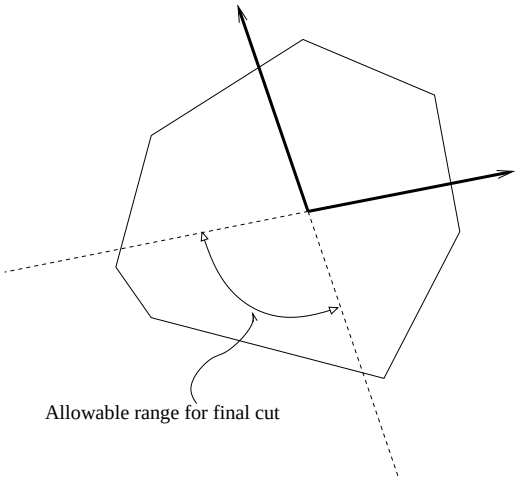

Fig. 5. Range constraints for pie cutting.

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.

3.6 The Final Heuristic

We formulate a method combining the operations of BSPs and Pie-Cuts. 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.

3.7 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 [Har-Peled 2004].

4. EXPERIMENTAL SETUP

Implementation of our system GeoSect 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. The GeoSect software is available online at the url `http://voronoi.ams.sunysb.edu/~gk/projects/GeoSect`.

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. See Figure 1. 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.

5. EXPERIMENTAL RESULTS

5.1 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 6. The selection of the regions was based on a visual inspection of the track data to correspond to both high and low traffic regions.

5.1.1 *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 6, 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.

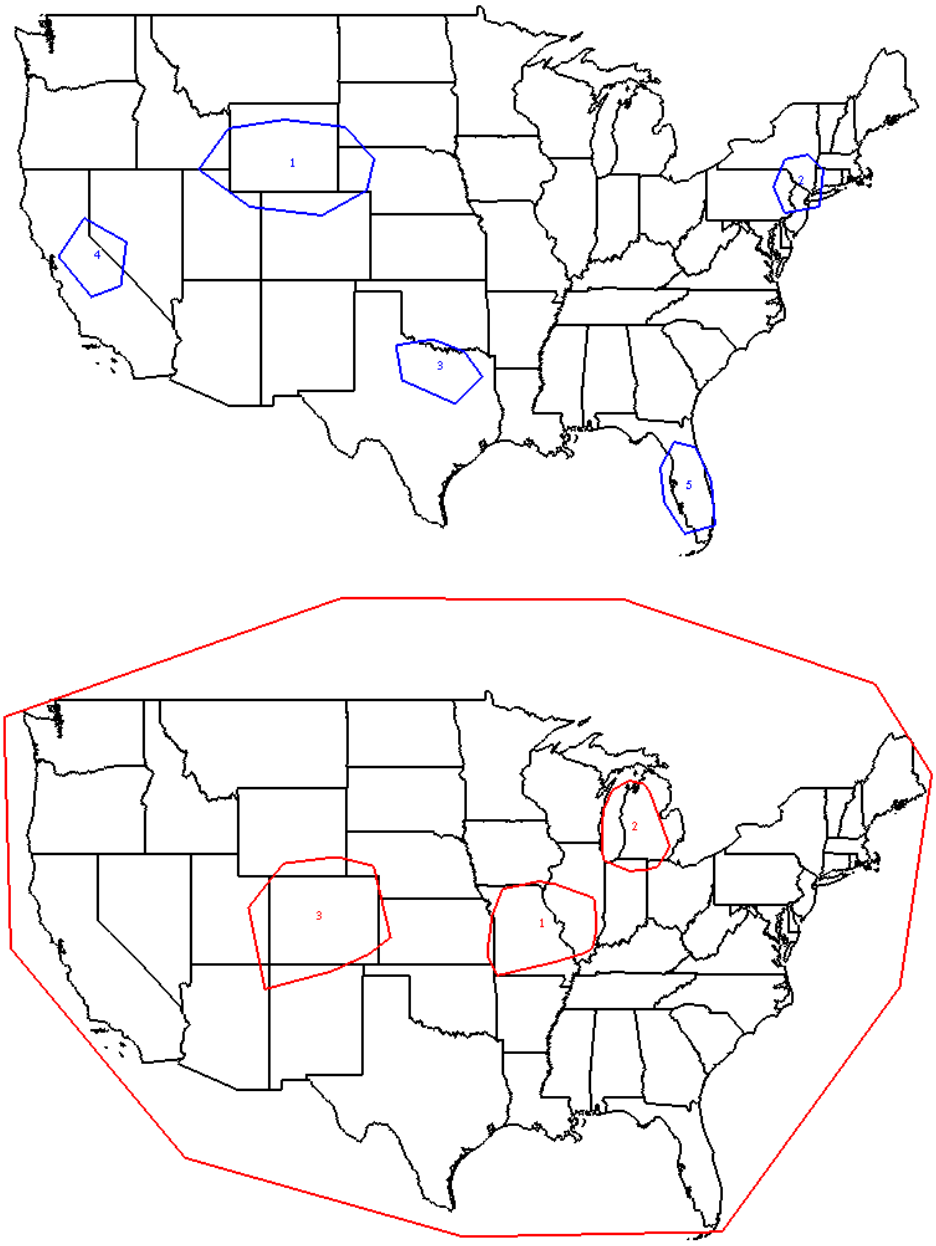

Fig. 6. Top: Regions used for tuning the parameters. Bottom: Regions used for testing the Final Heuristic.

—*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 (Figure 9) we show the effect of different choices of β on the final workload balancing.

Some results are presented in Figure 7. 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 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.

Region	$WorstcaseWorkload$		
	Max	$Avg.$	$Std.Dev.$
1	5	4.237	0.501
2	5	4.571	0.564
3	5	4.559	0.537
4	5	4.442	0.573
5	5	4.414	0.553

Region	$TimeAverageWorkload$		
	Max	$Avg.$	$Std.Dev.$
1	0.891	0.344	0.133
2	0.893	0.403	0.151
3	0.766	0.409	0.138
4	1.206	0.405	0.169
5	0.771	0.351	0.142

Region	α		
	$Avg.$	Min	$Std.Dev.$
1	0.572	0.350	0.139
2	0.609	0.350	0.156
3	0.593	0.350	0.148
4	0.591	0.350	0.141
5	0.587	0.350	0.149

Fig. 7. BSP results for the 5 regions after parameter tuning.

5.1.2 *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 ratios, as mentioned previously.

5.1.3 *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.

5.2 Achieving the Balancing

The main goal is to balance the time-average workload across all of the 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(the total workload divided by k), 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 the sectors is controlled, as described previously, by avoiding the cuts that result in bad aspect ratios. There definitely is a trade-off between preserving good aspect ratios and optimally balancing the sectors. This trade-off is indicated in the results in Figure 9.

5.3 Comparing the Final Heuristic with Existing Sector Data

In our comparisons of our Final Heuristic with the original sectors, experiments were conducted 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 6.

When computing statistics for the original sectors, we consider only the sectors that are completely inside the domain of interest. While we compute both time average workloads and worstcase workloads of the sectorizations we compute, the partitioning in each of the reported experiments was done by choosing cuts in order to balance the time-average workload.

5.3.1 *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 Figure 8. 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; thus, while the average workload for original sectors is computed on only those original sectors fully contained within the domain, the average workload of the sectors we compute includes all travel within the domain, since our sectors perfectly partition Domain 1. The BSP results are shown as well. This experimentally supports our claim that our heuristics achieve highly balanced workloads, while avoiding skinny sectors.

Sectorization	No. of Sectors	Time Average Workload		
		Max	*Avg.*	*Std.Dev.*
Original Sectors	10	12.33	6.899	2.578
Final Heuristic	10	7.289	7.226	0.054
BSP	10	7.9525	1.226	0.302

Sectorization	No. of Sectors	Worstcase Workload		
		Max	*Avg.*	*Std.Dev.*
Original Sectors	10	44	26.8	8.340
Final Heuristic	10	30	27.9	1.221
BSP	10	31	27	2.323

Sectorization	No. of Sectors	α		
		Avg.	*Min*	*Std.Dev.*
Original Sectors	10	0.548	0.264	0.169
Final Heuristic	10	0.534	0.323	0.159
BSP	10	0.522	0.25	0.171

Fig. 8. The statistics for pure BSP, the Final Heuristic and the original sectors for Domain 1.

5.3.2 *Results for "Domain 2".* As with Domain 1, in the Domain 2 (full 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 (except that for Pie-Cut it was 412, due to the nature of the combinatorics of Pie-Cut partitions). Results are tabulated in Figure 9.

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 ratios $\alpha \geq \beta$. (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

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

Fig. 9. 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.

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 ratios.

Refer to Figure 10 and Figure 11 for some screenshots of the sectorizations we compute.

5.4 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) [Yousefi 2005]. The MIP method considers the domain of interest to be a union of small regular 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 Figure 12 we tabulate the results of our comparison 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 13 for the screenshots of these comparison.

6. EXTENSIONS TO THE MODEL AND CONCLUSIONS

We have studied in detail the optimal sectorization problem that arises in air traffic management, giving a theoretical formulation, algorithmic results in the 1D setting and special cases of the 2D setting, and experimental results in the general 2D setting.

Current and future work is aimed at extending our model and the implementation to take into consideration more of the factors that directly influence workload complexity in quantifiable ways, such as the coordination workload (proportional to how many flights cross the boundary of a sector), altitude changes, anticipated conflict avoidance (to enforce separation standards), traffic mix (variety of aircraft), angles of intersection between crossing flows, variation in headings, location of "hot spots" (e.g., merge points) within sectors, holding patterns, etc (see [Wyndemere 1997] for more details). Already, GeoSect includes the option to optimize a linear combination of workload and coordination workload (which accounts for the number of times a flight must be "handed off" between controllers). Preliminary experiments show that, by optimizing a linear combination of time-average workload (weighted 0.7) and coordination workload (weight 0.3) we are still able to balance the the time-average workload while preserving similar coordination workloads as the original sectors.

Another important direction in which we have extended our model is to account

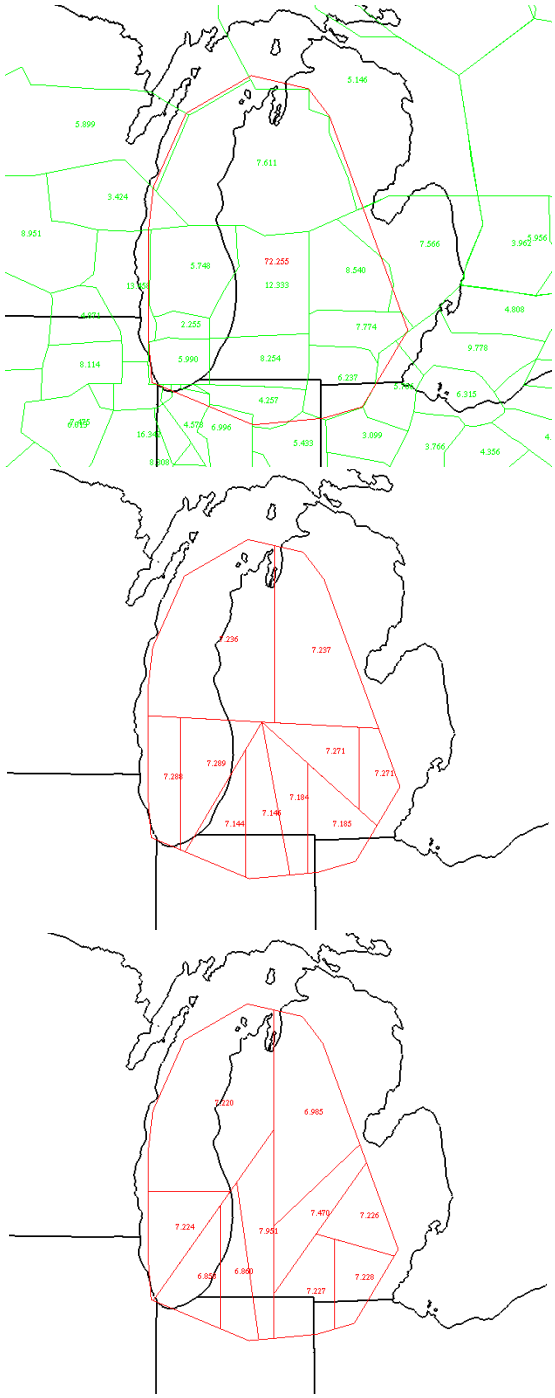

Fig. 10. Partition results for Domain 1. Top: Original sectors. Middle: Final Heuristic partition. Bottom: BSP.

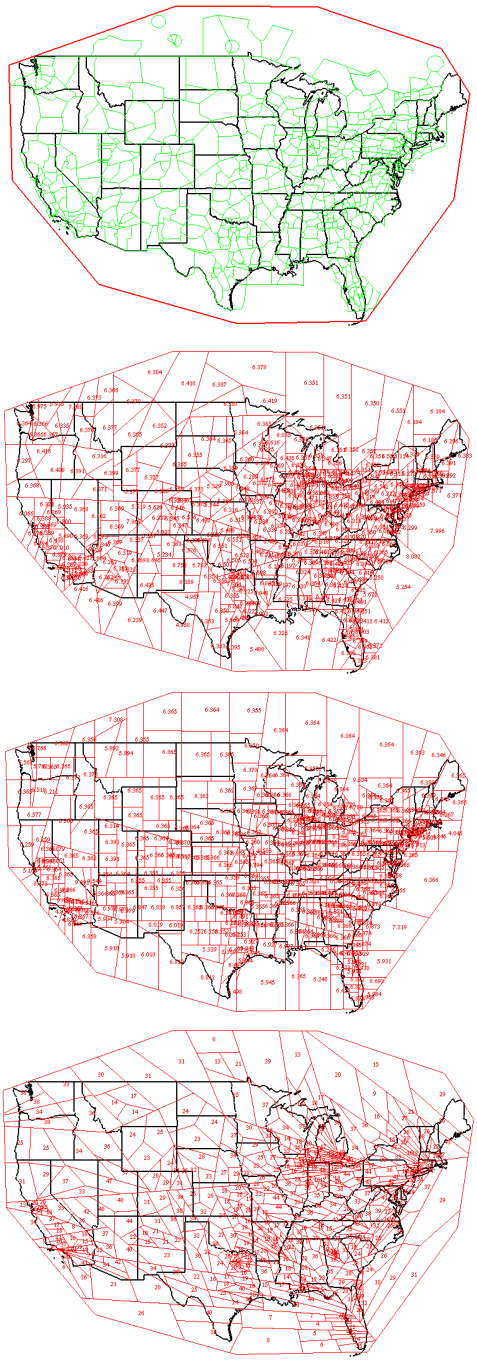

Fig. 11. Partition results for the continental USA (Domain 2). From Top (i) original sectors (411) strictly within the selected region, (ii) Final Heuristic partition (411 sectors), (iii) BSP (411 sectors), (iv) Pie-Cut partition (412 sectors).

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

Fig. 12. The statistics for the MIP solutions and the Final Heuristic for sectorizing ZFW (Dallas) center.

for constraints in the NAS that affect the shapes of sectors. The "odd" shapes of current sectors arise not only from historical artifacts but also partly from certain domain constraints on sector boundaries so that, e.g., they do not pass through certain regions of special use airspace (SUA) or pass too close to airports, etc. We have extended our model to include such constraints, allowing the partitioning to be done only with cuts that avoid specified constraints. In order to make a richer set of cuts available, we then also permit a cut to be a polygonal *chain* (based on a shortest constraint-avoiding path that has a limited number of bends) rather than a straight segment; thus, the sectors obtained no longer constitute a BSP, and sectors may be non-convex. We are implementing this feature into GeoSect for future experimentation; results will be reported elsewhere.

More investigation is needed on other track data too. We plan to run experiments with GeoSect on track data given by wind-optimized routing data. Future experiments will also address the more complex problem in the terminal area, for which generalizations to three dimensions will be necessary. In principle, our techniques apply to 3D sector design; we have, in fact, started to experiment with a 3D version of GeoSect, allowing cuts at constant altitude levels.

On the theoretical side, it would be interesting to investigate further provable approximation algorithms for the 2-dimensional sectorization problems studied here. Also, is it possible to give a deterministic $O(n \log n)$ algorithm to compute the lowest vertex of a k-level in an arrangement of lines or line segments?

ACKNOWLEDGMENTS

We thank an anonymous reviewer for helpful comments and corrections to an earlier draft. This work is supported under NASA Ames Research Center's Advanced Air Transportation Technologies (AATT) project under Task Order 20 for contract NNA04BA29T. We thank Parimal Kopardekar, Greg Wong, and Shannon Zelinski of NASA Ames for guidance and technical input. We thank Jimmy Krozel,

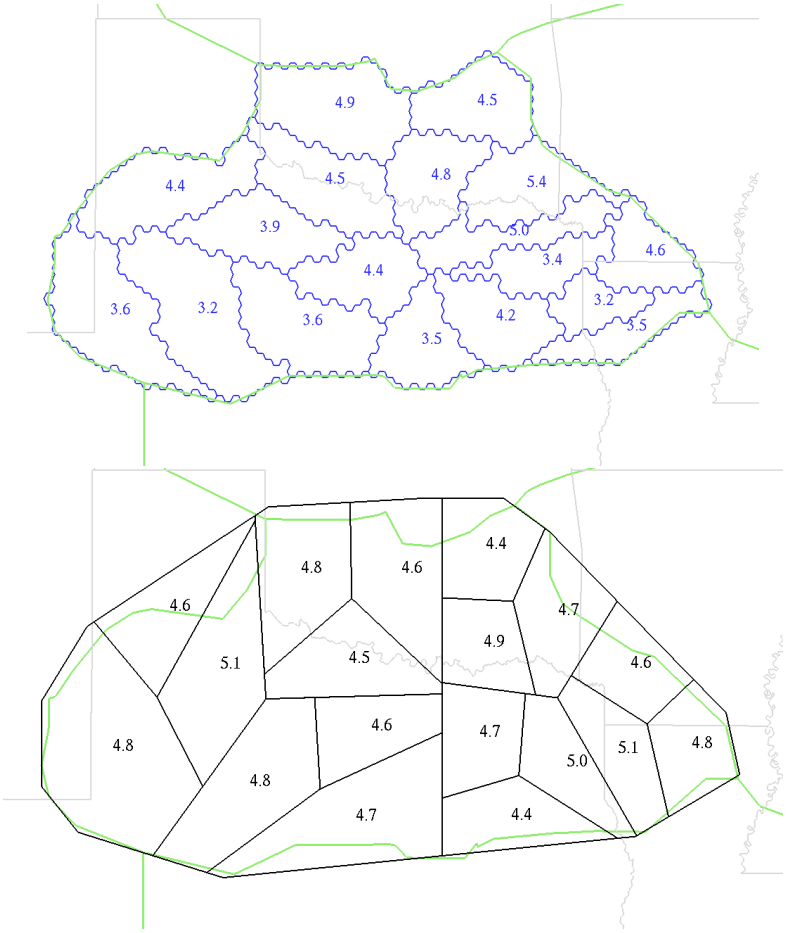

Fig. 13. Partition results for ZFW Center. Top: MIP method. Bottom: Final Heuristic partition.

Arash Yousefi, Bob Hoffman, and Terry Thompson of Metron Aviation for useful discussions and domain expertise on air traffic management. We also thank Metron Aviation for providing us with the sector data and historical track data.

REFERENCES

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

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

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

Berman, P., DasGupta, B., and Muthukrishnan, S. 2000. On the exact size of the binary space partitioning of sets of isothetic rectangles with applications. Dimacs report.

Berman, P., DasGupta, B., and Muthukrishnan, S. 2002. Slice and dice: A simple, improved

approximate tiling recipe. In *Proc. 13th ACM-SIAM Symposium on Discrete Algorithms*. 455–464.

Berman, P., DasGupta, B., Muthukrishnan, S., and Ramaswami, S. 2001. Improved approximation algorithms for rectangle tiling and packing. In *Proc. 12th ACM-SIAM Symposium on Discrete Algorithms*. 427–436.

Carmi, P. and Katz, M. J. 2005. Minimum-cost load-balancing partitions. In *Proc. 17th Canadian Conference on Computational Geometry*. 63–65.

Chan, T. M. 1999. Geometric applications of a randomized optimization technique. *Discrete and Computational Geometry 22,* 4, 547–567.

Chan, T. M. 2006. Personal communication.

Cole, R., Salowe, J., Steiger, W., and Szemerédi, E. 1989. An optimal-time algorithm for slope selection. *SIAM Journal on Computing 18,* 4, 792–810.

Delahaye, D., Schoenauer, M., and Alliot, J. M. 1998. Airspace sectoring by evolutionary computation. In *Proc. IEEE International Congress on Evolutionary Computation*.

Forman, S. L. and Yue, Y. 2003. Congressional districting using a TSP-based genetic algorithm. In *Proc. Genetic and Evolutionary Computation Conference*. Lecture Notes in Computer Science, vol. 2724. Springer-Verlag, 2072–2083.

Har-Peled, S. 2004. Clustering motion. *Discrete and Computational Geometry 31,* 4, 545–565.

Hendy, K. C., Liao, J., and Milgram, P. 1997. Combining time and intensity effects in assessing operator information and processing load. *Journal of Human Factors*, 30–47.

Khanna, S., Muthukrishnan, S., and Paterson, M. 1998. On approximating rectangle tiling and packing. In *Proc. 9th ACM-SIAM Symposium on Discrete Algorithms*. 384–393.

Khanna, S., Muthukrishnan, S., and Skiena, S. 1997. Efficient array partitioning. In *Proc. International Colloquium on Automata, Languages and Programming*. 616–626.

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

Muthukrishnan, S., Poosala, V., and Suel, T. 1999. On rectangular partitionings in two dimensions: Algorithms, complexity, and applications. In *Proc. 7th International Conference on Database Theory*. Lecture Notes in Computer Science, vol. 1540. 236–256.

Muthukrishnan, S. and Suel, T. 2005. Approximation algorithms for array partitioning problems. *Journal of Algorithms 54,* 1, 85–104.

National Airspace Redesign (NAR). National airspace redesign (NAR). Office of Air Traffic and Airspace Management, Federal Aviation Administration, http://www.faa.gov/ats/nar/.

Occupational Outlook Handbook. Occupational outlook handbook. Bureau of Labor Statistics, U.S. Department of Labor, http://stats.bls.gov/oco/.

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

Paterson, M. S. and Yao, F. F. 1992. Optimal binary space partitions for orthogonal objects. *Journal of Algorithms 13*, 99–113.

Schmidt, D. K. 1976. On modeling ATC work load and sector capacity. *Journal of Aircraft 13,* 7, 531–537.

Stein, E. S. 1998. Human operator or load on air traffic control. In *Human factors in air traffic control*, M. W. Somlensky and E. S. Stein, Eds. Academic Press, 155–183.

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

Tran, D. H., Baptiste, P., and Duong, V. 2003. Optimized sectorization of airspace with constraints. In *Proc. 5th FAA and EUROCONTROL ATM Conference*. Budapest, Hungary.

Wyndemere, I. 1997. Dynamic resectorization: Accommodating increased flight flexibility. Technical report, http://www.wynde.com/papers/atca97-2.pdf, Boulder, CO.

YOUSEFI, A. 2005. Optimum airspace design with air traffic controller workload-based partitioning. Ph.D. thesis, George Mason University.

YOUSEFI, A. AND DONOHUE, G. L. 2004. Temporal and spatial distribution of airspace complexity for air traffic controller workload-based sectorization. In *Proc. AIAA 4th Aviation Technology, Integration and Operations (ATIO) Forum*. Chicago, Illinois.

Received xxx; xxx; accepted xxx