Appeared in Proc. 36th IEEE Conference on Decision and Control (1997)

Geometric Algorithms for Conflict Detection/Resolution in Air Traffic Management[1]

Yi-Jen Chiang James T. Klosowski Changkil Lee
Joseph S. B. Mitchell
Department of Applied Mathematics and Statistics
State University of New York, Stony Brook, NY 11794-3600
{yjc,jklosow,blessed,jsbm}@ams.sunysb.edu

Abstract

We consider the problems of conflict detection and resolution in air traffic management (ATM) from the perspective of computational geometry and give algorithms for solving these problems efficiently. For conflict resolution, we propose a simple method that can route multiple aircraft, conflict-free, through a cluttered airspace, using a prioritized routing scheme in space-time. Our algorithms have been implemented into a simulation system that tracks a large set of flights, having multiple conflicts, and proposes modified routes to resolve them. We report on the preliminary results from an extensive set of experiments that are under way to determine the effectiveness of our methods.

1 Introduction

The FAA is considering a shift from the current air traffic control (ATC) system to a new system of air traffic management (ATM) based on a policy known as "Free Flight" [11]. In the current ATC system, flight paths require aircraft to travel set route segments, flight corridors in the "highway in the sky." In Free Flight, considerably more autonomy will be granted to individual aircraft, using a more passive control mechanism, with controllers intervening only in exceptional cases. The potential benefits of such a system are considerable, since it would allow aircraft to fly more fuel-efficient routes, potentially saving millions of dollars each year.

Critical to the safety of such a system is the use of instrumentation and software to predict potential *conflicts* between aircraft *before* they occur, and to plan negotiated alternative routes to *resolve* any such conflicts. Typically, a "conflict" is considered to occur when two aircraft that occupy approximately the same altitude (within about 1000 feet of one another) come within a distance of about 5 nautical miles of each other.

[1]This work was supported in part by Hughes Research Laboratories and NSF Grant CCR-9504192.

Our approach to the conflict prediction/resolution problems is to cast them into a precise geometric framework, in order to devise algorithms based on the methodologies of computational geometry. Here, we present some of our algorithm design work and report on some recent experimentation with a simulation system that implements some of our algorithms.

Related Work: There has been considerable previous work on conflict prediction and resolution; we do not attempt to survey it here, but refer the reader to the recent report of Krozel, Peters, and Hunter [18]. Briefly, we refer to the work of Cross [7] and Davis [8], who considered knowledge-based system approaches, the work of Eby [9], who considered a potential fields method of resolution, the work of Wangermann and Stengel [23], who considered agent-based and principled negotiation methods, and the work of Krozel, Mueller, and Hunter [17], who used optimal control theory to devise tactical alert zones to identify control strategies to prevent conflicts. Chen et al. [6] have applied network routing models (the dynamic network flow problem) to perform joint routing of aircraft, in order to minimize the total cost of time-disjoint routes.

Inselberg [15] has identified conflict resolution as a problem in computational geometry, relating it to the "asteroid avoidance problem" that was studied by Reif and Sharir [21]. Chen, Hsieh, Inselberg, and Lee [5] have proposed a method of conflict resolution related to our own, based on the heuristic of adding aircraft's trajectories one by one to a workspace, and searching for feasible routings that preserve clearances determined by the protected air spaces. Below, we cite some other relevant computational geometry results.

2 Preliminaries

Figure 1 shows a schematic diagram of a TRACON (Terminal Radar Approach Control) center, having four "gates" for aircraft to enter/leave, and two runways. Also shown are two "No-Fly Zones".

We model the airspace as a box in 3-space, having a discrete set of points (representing TRACON gates) and a set of no-fly zones, each given by a simple polygonal boundary together with a pair of delimiting altitudes. We model aircraft as points moving in space, under dynamic constraints that impose bounds on speed and acceleration, including climb/descent rates and turning radius. We let $p_i(t)$ and $v_i(t)$ denote the position and velocity of aircraft i at time t. All coordinates are relative to a global Cartesian coordinate system.

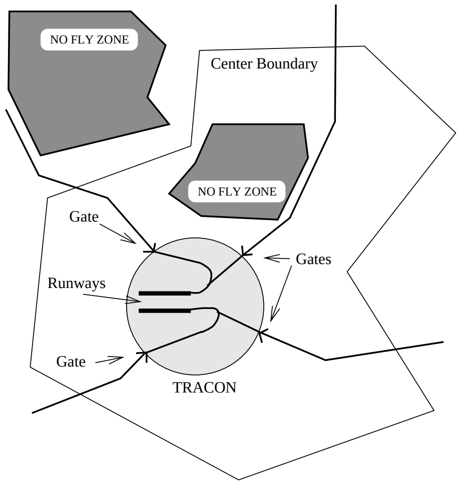

Figure 1: The geometry of the airspace and arrival routes to two runways within one TRACON.

Delaunay and Voronoi Diagrams: (see [20, 10]) The *Delaunay diagram* (DD) of a discrete set of points (*sites*) in the plane joins two points with an edge (line segment) if and only if there exists a circle through the two points, having no other points in its interior or on its boundary. The *dual* of a Delaunay diagram in the plane is a planar subdivision known as the *Voronoi diagram*, which partitions the plane into regions (*Voronoi cells*) according to which site is closest. If no four sites are cocircular, the Delaunay diagram is a triangulation; otherwise, it can readily be completed to yield a triangulation. In the plane, the Delaunay diagram for n points is a planar graph having $O(n)$ edges; it can be computed in $O(n \log n)$ time.

The definitions of Delaunay and Voronoi diagram extend naturally in many ways, to higher dimensions, other metrics, etc. In three dimensions, the DD can be computed in time $O(n^2)$, which is worst-case optimal, since there are sets of n points in $\Re^3$ for which the DD has $\Omega(n^2)$ edges.

The *nearest neighbor graph* (NNG) of a set of points (sites) is a graph whose edges join each site to the site(s) that is(are) closest to it. It is well known that the NNG is a subgraph of the Delaunay diagram. Thus, by computing the Delaunay diagram of points in the plane, we are able to compute all of the nearest-neighbor information efficiently, in time $O(n \log n)$, versus the naive all-pairs bound of $O(n^2)$.

3 Conflict Prediction

We say that a *conflict* occurs between aircraft i and j if one of them enters the *Protected Airspace Zone* (PAZ) of the other one. The PAZ of an aircraft is typically chosen to be a vertical cylinder centered on the aircraft, having a circular base of radius r_p (typically 5 nautical miles) and a height of h_p (typically 2000 feet).

In order to determine if a conflict occurs during some time window $t \in [0, T]$, we must determine, for every pair of aircraft (i and j), if there exists a moment at which the two are in conflict with one another. A straightforward approach is to examine all $O(n^2)$ pairs of aircraft: For aircraft i and j, consider the trajectory $p_{i,j}(t) = p_j(t) - p_i(t)$ that gives the location of aircraft j relative to a coordinate system centered at aircraft i, and determine if the curve $\{p_{i,j}(t) \mid t \in [0, T]\}$ intersects a copy of the PAZ, centered at the origin (assuming that the PAZ is the same for aircraft i and j). If the curve intersects the PAZ, then there is a conflict, and the first moment of conflict can be computed.

3.1 A Delaunay Approach

Since Delaunay diagrams succinctly encode nearest neighbor information, a natural approach to increasing the efficiency of conflict prediction is to maintain the Delaunay diagram of the aircraft over time. Then, instead of monitoring all pairs of distances, $d_{i,j}(t) = d(p_i(t), p_j(t))$, we only have to determine if some edge of the Delaunay diagram ever becomes shorter than the radius r_p of the PAZ.

Thus, it is useful to study the evolution of the Delaunay diagram, $DD(t)$, of a set of moving sites, over time. This problem has been studied, both in the plane ([12, 14, 22]) and in higher dimensions ([1]). In particular, Guibas et al. [14] have given an efficient algorithm to maintain and update the Delaunay diagram over time, for points moving on fixed trajectories. The algorithm maintains a priority queue of *events* that correspond to four points becoming cocircular, when the topological structure of $DD(t)$ changes. Their algorithm spends optimal $O(\log n)$ time per event, and they prove a bound of roughly $O(n^3)$ on the worst-case number of events for points moving along well-behaved (e.g., bounded algebraic degree) trajectories.

We have implemented (in C) a conflict prediction system, based on the algorithm of [14]; the core Delaunay algorithm was implemented by Gerhard Albers. We currently assume that the input is a set of n points (aircraft), each moving along piecewise-linear trajectories, encoded as a sequence of vertices (*way points*).

By maintaining $DD(t)$ over a time horizon $t \in [0, T]$, we are able to report pairs of aircraft that come into conflict, and we identify clusters of aircraft in potential conflict in their neighborhood (within $DD(t)$).

There are some advantages to the Delaunay approach. First, it incorporates a precise notion of separation (protected airspace), allowing one to reason about other proximity issues and to identify clusters of aircraft that may be involved in conflicts if we re-route some aircraft. Also, it is efficient, avoiding an *all-pairs* calculation and spending time $O(\log n)$ *per event*. Further, the method allows one to check *only* those potential conflicts that can arise during a specified look-ahead time window. Finally, the approach is applicable to other metrics and higher dimensions.

In recent work, related to the Delaunay approach, Basch et al. [2, 3] have proposed a new set of "kinetic data structures" designed to maintain efficiently the *closest pair* (among the $\binom{n}{2}$ possibilities) for n points in continuous motion in the plane. (See [4] for results in $\Re^d$.)

3.2 A Simple Geometric Hashing Approach

While the Delaunay approach has some advantages and a theoretical basis, it also has limitations: the need to solve high-degree equations (degree $4k$ for trajectories of degree k), which is prone to numerical errors, and the complexity of implementing it in 3-space. Thus, for our simulations, we have been using a simple geometric hashing approach, based on discretizing (tiling) the airspace with boxes whose size corresponds to the PAZ (length and width $2r_p$, and height h_p). We also partition the given time window into discrete time steps of size Δt. Our goal, then, is to identify the set of all aircraft that are in conflict with some other aircraft at any one of these time steps in the time window. Because of our choice of grid box, we only need to check, for each aircraft i lying in grid box b_i, all aircraft j lying in the 27 neighboring boxes of b_i, including b_i itself. In practice, most boxes are empty or contain only a small number of aircraft; thus, this test tends to take time linear in n.

For a given look-ahead time window W, our algorithm identifies subsets of aircraft in potential conflict; these are then passed into the resolution algorithm, so the goal is to keep the subsets small, while still being conservative. For this purpose, we report subsets that are non-singleton connected components in the *conflict graph*, G, whose nodes correspond to aircraft and whose edges join pairs of nodes whose corresponding aircraft are in conflict at any time during W. (The connected components are found using depth-first search in G.)

In addition to reporting each conflict cluster to the resolution algorithm, it is important that we also identify those *non*-conflict aircraft (called *buffer aircraft*) that are sufficiently close to a given cluster, in order that these can be treated as constraints ("obstacles") during resolution of that cluster; i.e., we want the resolution algorithm to treat the buffer aircraft as constraints, and not to create new conflicts with them (and thereby create a "domino effect"). Again, our goal is to keep the number of buffer aircraft small. We identify the buffer aircraft associated with a cluster C_i as the set of aircraft that, during time window W, enter the slightly enlarged (by a fixed factor) axis-aligned bounding box, B_i, of the trajectories of the cluster aircraft (within W). The box B_i serves as the airspace in the resolution algorithm. We apply the resolution algorithm to each cluster *independently*, according to the order in which we consider clusters (which can be varied, in order to account for different priorities, or in order to search more aggressively for resolutions). Observe that airspace B_i of the current cluster C_i may overlap with another cluster's airspace, B_j, causing dependencies among clusters. We address this issue by considering all aircraft from previously resolved clusters whose *new* paths enter B_i to be *additional* buffer aircraft associated with C_i.

4 Conflict Resolution

The input to the resolution algorithm is a set of aircraft (points), with initial ($t = 0$) positions and velocities, and a set of destination locations and velocities. There is also a time window associated with each aircraft's destination, to reflect the constraint that it reach its destination roughly on schedule. Each of the input aircraft also has associated constraints on velocity (maximum and minimum speed) and acceleration (determining, e.g., minimum turning radius, maximum climb rate, etc.)

The output is either a statement that no resolution was found (subject to the given search parameters), or a set of trajectories, specified by a list of *way points*, for each aircraft, obeying all dynamic and separation constraints (avoiding conflicts).

The problem of conflict resolution is closely related to some problems in motion planning in the presence of moving obstacles [5, 13, 15, 21]. In general, these problems are known to be NP-hard [21].

Thus, our approach is to examine efficient algorithms, based on selective discretizations, that are simple to implement, while providing provable resolutions to all conflicts – i.e., it is guaranteed that resolutions proposed by our algorithms do in fact resolve conflicts. Note, however, that it is possible that our algorithm reports that no feasible resolution exists, when in fact one does; in this sense, our algorithm is not what is formally called "complete." (We are not claiming to solve an NP-hard problem in polynomial time!) However, our algorithm does "degrade gracefully", taking a longer time for more complex situations, while being very fast for the vast majority of simple conflict scenarios. This is achieved by iteratively adjusting the search parameters that control the degree of discretization. (In the limit, as the discretization becomes finer, the search algorithm will converge on an exact decision procedure for the NP-hard problem, but the running time will increase substantially as we approach this limit.)

We view the problem as that of finding a feasible set of *tubes* ("pipes") through space-time, each of which corresponds to the trajectory of an aircraft, subject to dynamics constraints. Each tube represents the portion of space-time occupied by the PAZ of an aircraft along its trajectory. This is easiest visualized in the case of 2-dimensional PAZs, so that space-time is 3-dimensional, with the tubes being seen as "pipes" whose radii (in any time slice of space-time) represent the separation required between conflict-free aircraft. The tubes also have a similar interpretation in 4-dimensional space-time, with a time-slice corresponding to a PAZ (e.g., "hockey puck") at one position of the aircraft on its trajectory. The problem is formally a type of constrained "flow" problem in a continuous space. (Such continuous flow problems have been studied algorithmically by Mitchell [19].)

Space-Time Flow (STF) Method: Our method is based on an iterative procedure for adding tubes (aircraft), one-by-one, using a graph search in discretized space-time to route each tube amongst the already routed tubes, which are considered to be obstacles. (A set of buffer aircraft are also considered to be obstacles; see Section 3.) Specifically, the look-ahead time window is discretized into a number, N, of time slices, where N is a parameter that is adjusted adaptively in the search for a feasible routing of each flight. Initially, $N = 0$, and then N is increased, incrementally or by doubling, up to a maximum allowed value ($\overline{N}$), until a feasible routing is found. Further, for each time slice, we also discretize the problem spatially, either by (a) imposing a grid (with spacings Δ_x, Δ_y, and Δ_z), or by (b) discretizing the allowable heading changes for each aircraft.

A candidate (straight) edge linking two points of space-time is *feasible* if and only if the corresponding tube determined by sweeping the PAZ along the edge does not intersect already routed flight paths.

We search the graph of feasible linkages for a path to route the current tube, subject to the dynamics constraints (computed on a link-by-link basis during the search) and the constraints imposed by tubes of already routed aircraft. We use two different search methods: one based on breadth-first search (BFS – essentially Dijkstra's shortest path algorithm), and one based on depth-first search (DFS). The BFS method uses the grid discretization ((a), above), in order to prevent explosion in the total size of the graph; the DFS method uses a discretization of the heading changes ((b), above), in order to control the degree of the graph being searched.

The order in which we consider aircraft for routing is important and can determine not only the solution obtained, but also the success or failure of the search for a feasible routing. One aspect of this iterative nature of the algorithm, though, is that it permits one to impose a *priority* on the set of aircraft, since the aircraft that appear earlier in the ordering are more likely to be routed using a fuel-efficient, more direct path. Thus, we can utilize this feature to our advantage, e.g., if there are preferences among types or sizes of aircraft, this can determine the ordering. Another option is to use a "greedy heuristic" in the ordering: Select the aircraft to route next based on which one results in a route having the least (or possibly the greatest) deviation from its optimal (or direct) path. But we can also attempt to search over many, or all, possible orderings, in order to be most aggressive at resolving conflicts; this is in fact the default in our simulation system.

A critical computational primitive in our algorithm is the test of feasibility of a graph edge, wherein we must test for intersection between a candidate tube (in space-time) and the set of existing trajectory segments. For this primitive, we are currently performing a "brute force" test against all existing segments. (Indeed, this is one reason to keep the size of clusters and the number of buffer aircraft small.) However, we have also been experimenting with the possible use of "BV-trees" to speed up this query; see Section 6. BV-trees are the basis of highly efficient intersection searches, within the *QuickCD* system developed by Klosowski et al. [16] for motion simulation in virtual environments.

Since our algorithm only searches for routes that are feasible, subject to the dynamic constraints and the PAZ of each aircraft, we are *guaranteed* that the routes we produce do not have conflicts. (This is not true of the routes produced by some other methods, e.g., the "self-organizational" approach of Eby [9].) Also, the algorithm naturally allows alternative prioritization schemes to be able to suggest alternative resolutions, which can be presented to humans in the loop specified by pilots or controllers. Further, the method can be applied to account explicitly for the incorporation of feeder gateways at TRACON regions, by putting time windows accordingly for flights that terminate at each feeder gateway. This modeling feature can be used to enforce aircraft to line up, with specified separations, as they enter and leave TRACON regions. Finally, our algorithm readily handles no-fly zone constraints, since each no-fly zone corresponds to a space-time obstacle whose time slices are the no-fly zone regions.

5 System Overview

We have implemented a simulation environment to test our implemented algorithms for conflict prediction (the CP module) and conflict resolution (the CR module).

The simulation is conducted over a time horizon, $t \in [0, \overline{T}]$. We discretize the horizon into small time steps (of size ΔT minutes). We perform conflict prediction over a *look-ahead window*, $W = [\tau, \tau + T]$, from the current time step, τ, over the next T minutes. The CP module detects any conflicts that will take place during W, identifies conflict clusters C_i, and passes them, one by one, along with their associated sub-airspace B_i and buffer aircraft, to the CR module. The CR module attempts to compute a resolution for each cluster (within the time window W), and, if successful, returns a set of feasible trajectories that get the aircraft through the look-ahead window. The system then uses these trajectories to update the (global) routes, and the CP module is then called again for the updated routes, after advancing the look-ahead window forward by ΔT.

One issue that we must address is that the interpolated destination locations of the aircraft at the end of the look-ahead window ($t = \tau + T$) may themselves be in conflict, making it impossible for the CR module to reach a resolution. To address this, in such cases, the system searches forward from $\tau + T$, in order to find a time t for which there is no conflict (for all aircraft).

6 Experimental Results

We have conducted a first series of experiments based on datasets using real (airport location) and simulated (flight path) data. Our goal was to devise challenging conflict situations to test our algorithms.

We used a set of 60 actual airport locations, spanning a region 1186 km by 1132 km, in California, Nevada, and Arizona. We generated a large collection of simulated flights, between random pairs of airports (having separation distance at least 500 km), with departure and arrival times that fall within the time horizon of $\overline{T} = 10$ hours, subject to the speed constraints on aircraft. All aircraft had speed 900 (km/hr) at cruising altitude (10 km), and had (ground) speed 450 (km/hr) during ascent and descent (at ascent/descent rate 40 km/hr). We used a minimum turning radius of 5 km. We constructed routes for each flight, one by one, by searching for a feasible (with respect to flights already constructed) trajectory, using the resolution algorithm. If a generated flight could not be feasibly routed, it was discarded and a new one generated. This process continued until we had placed as many flights within the airspace and time horizon as possible, stopping when we encountered 100 consecutive failed attempts to route randomly generated flights. For the resulting 858 successfully routed flights, we then "erased" the trajectories (whose existence is a proof of feasibility), and recorded only the origin and destination data, for each flight. This set of flights constitutes a "saturated" set of flights. We then ran our simulation on different fractions ($\sigma = 30, 50\%$) of the flights, chosen at random from the set.

For the data generation, we used an upper bound of $\overline{N} = 4$ on the number of allowed way points in a resolution within W; however, for the experiments, we studied the effect of varying $\overline{N}$, for $\overline{N} = 4, 8, 16$. (When a choice of number of way points, N, failed to find a route, we doubled N, as long as $N \leq \overline{N}$.)

We used a time step of $\Delta t = 0.1$ minutes in the CP module, and a look-ahead window of length $T = 30$ minutes, which we advance by $\Delta T = 5$ minutes after each call to the CR module. We used a minimum turning radius of 3 km, and considered heading changes in 1-degree increments, from 35-degrees left, to 35-degrees right.

We recorded several statistics in our experiments. Here, in the table below, we report two of the most relevant numbers: the resolution *success rate*, ρ, and the CPU time (in milliseconds) required by the CR module, *per resolved aircraft*. We define ρ to be the ratio of the average number of flights resolved to the average number of flights in a conflict component, averaged over all calls to the CR module (for each look-ahead window during the simulation of the 10 hours of flights).

		Max # Way Points, $\overline{N}$					
		4		8		16	
Search	σ	ρ	ms	ρ	ms	ρ	ms
BFS	30	.969	11.9	.969	11.6	.969	12.3
BFS	50	.943	32.8	.943	33.5	.943	33.6
DFS	30	.926	12.7	.926	12.1	.926	12.7
DFS	50	.855	37.1	.868	631.6	.868	1214.9

For each cluster C_i that is passed to the CR module, we generate and try *all* possible permutations of aircraft within C_i. (However, we update the global routes using only one feasible resolution – the first one found.) Cluster sizes range from 2 to 3, with an average of 2.1. In contrast, the average number of aircraft in conflict over W was 2.2 (ranging from 2 to 8), and the number of "obstacle" aircraft averaged 11.2 (from 2 to 29).

We also generated a saturated dataset based on a larger threshold (500 vs. 100) on the number of consecutive failed attempts to route a random flight. Naturally, this resulted in more flights (1669 vs. 858) being successfully routed. While our resolution code was not able to resolve the set of flights (using the parameter settings above), it was interesting to see that the component sizes sent to the CR module varied only from 2 to 6, while the number of aircraft in conflict (during W) varied from 2 to 57 (with an average of 12).

Finally, we have also experimented with the impact of using BV-trees in possibly speeding up the CR module. For this purpose, we devised a simple experiment in which we randomly generated 100 disjoint tubes in a box (workspace) in 3D, and then performed intersection queries with randomly generated tubes (meant to correspond to feasibility tests for candidate edges during the CR search). We found that BV-trees provided a speedup of roughly a factor of 2-4 over the brute-force comparison against all tubes, as is currently done in the CR module. In order to integrate this method into the system, though, the BV-tree algorithm must first be made "dynamic" so that new tubes can be added.

Acknowledgments: We thank Ron Azuma and Mike Daily of Hughes Research Labs and Jimmy Krozel of Seagull Technologies, for introducing us to these problems and for many useful discussions about this research. We thank G. Albers, P. Remmele, and T. Roos for providing their implementation of the algorithm of [14].

References

[1] G. Albers, L. Guibas, J. Mitchell, and T. Roos. Voronoi diagrams of moving points. *Internat. J. Comput. Geom. Appl.*, to appear.

[2] J. Basch, L. Guibas, and J. Hershberger. Data structures for mobile data. In *Proc. 8th ACM-SIAM Sympos. Discrete Algorithms*, pages 747–756, 1997.

[3] J. Basch, L. Guibas, C. Silverstein, and L. Zhang. A practical evaluation of kinetic data structures. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 388–390, 1997.

[4] J. Basch, L. Guibas, and L. Zhang. Proximity problems on moving points. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 344–351, 1997.

[5] Y.-B. Chen, M. Hsieh, A. Inselberg, and H. Q. Lee. Planar conflict resolution for air traffic control. In *Proc. 2nd Canad. Conf. Comput. Geom.*, pages 160–163, 1990.

[6] Z. Chen, A. T. Holle, B. M. E. Moret, J. Saia, and A. Boroujerdi. Network routing models applied to aircraft routing problems. In *Proc. Winter Simulation Conf., Arlington*, pages 1200–1206, 1995.

[7] S. E. Cross. *Qualitative Reasoning in an Expert System Framework.* Ph.D. thesis, Univ. of Illinois at Urbana-Champaign, Urbana, IL, May 1983.

[8] G. A. Davis. AERA2 APR knowledge base. Technical Report MTR-89W00120, The MITRE Corp., McLean, VA, 1992.

[9] M. S. Eby. A self-organizational approach for resolving air traffic conflicts. *The Lincoln Laboratory Journal*, 7(2):239–254, 1994.

[10] S. Fortune. Voronoi diagrams and Delaunay triangulations. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 20, pages 377–388. CRC Press LLC, 1997.

[11] `http://www.faa.gov/asd/ffmain.htm`, Free Flight Home Page, FAA.

[12] J.-J. Fu and R. C. T. Lee. Voronoi diagrams of moving points in the plane. *Internat. J. Comput. Geom. Appl.*, 1(1):23–32, 1991.

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

[14] L. Guibas, J. S. B. Mitchell, and T. Roos. Voronoi diagrams of moving points in the plane. In *Proc. 17th Internat. Workshop Graph-Theoret. Concepts Comput. Sci.*, Springer-Verlag LNCS Vol. 570, pages 113–125, 1991.

[15] A. Inselberg. Conflict resolution, one-shot problem, and air traffic control. In *Abstracts 1st Canad. Conf. Comput. Geom.*, page 26, 1989.

[16] J. Klosowski, H. Held, J. S. B. Mitchell, K. Zikan, and H. Sowizral. Efficient collision detection using bounding volume hierarchies of k-DOPs. *IEEE Trans. Visualizat. Comput. Graph.*, to appear.

[17] J. Krozel, T. Mueller, and G. Hunter. Free flight conflict detection and resolution analysis. In *Proc. AIAA Guidance, Navigation, and Control Conference, San Diego*, 1996.

[18] J. Krozel, M. E. Peters, and G. Hunter. Conflict detection and resolution for future air transportation management. Technical Report TR-97138-01, Seagull Technology, Inc., Los Gatos, CA, April 1997.

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

[20] A. Okabe, B. Boots, and K. Sugihara. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams.* John Wiley & Sons, Chichester, UK, 1992.

[21] J. H. Reif and M. Sharir. Motion planning in the presence of moving obstacles. In *Proc. 26th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 144–154, 1985.

[22] T. Roos. Voronoi diagrams over dynamic scenes. *Discrete Appl. Math.*, 43:243–259, 1993.

[23] J. P. Wangermann and R. F. Stengel. Principled negotiation between intelligent agents: A model for air traffic management. In *Proc. 19th ICAS Congress*, Anaheim, CA, Sept. 1994.