Proceedings of the American Control Conference
Anchorage, AK May 8-10, 2002

Flying Hot Potatoes

Pradyumna Mishra[1] and George J. Pappas[2]
[1] Department of Computer and Information Sciences
[2] Department of Electrical Engineering
University of Pennsylvania
Philadelphia, PA 19104
{pmishra,pappasg}@grasp.cis.upenn.edu

Abstract

Optical communication networks and air traffic management systems share the same fundamental routing problem as both optical packets and aircraft must continuously move within the network, while avoiding conflicts. In this paper, we explore the use of hot potato and deflection routing algorithms, which are established routing methods in optical communication networks, in the conflict-free routing of air traffic. Hot potato algorithms allow the incorporation of conflict resolution constraints into the routing problem, in contrast to most approaches that decouple the optimal routing problem from the conflict resolution problem.

1 Introduction

Research in air traffic management systems has received much attention recently [8]. In particular, there has been much interest on conflict resolution algorithms [12, 6, 10, 5] as well as scheduling methods for traffic throughput maximization [11, 7, 3]. Despite considerable progress in both conflict resolution and optimal routing, the two areas of research remain mostly disconnected as routing methods do not incorporate conflicts inherently in their problem formulation.

In this paper, we take a step towards bringing optimal routing and conflict resolution closer together. In particular, we are inspired from routing methods in optical communication networks, where optical packets cannot be buffered and must therefore continuously move within the network without colliding. The *non-stationarity* of both optical packets and aircraft makes their routing problems very similar, compared to other routing problems encountered in computer networks, automotive networks, or robotics.

Hot potato algorithms [2, 1, 9] are established routing algorithms for optical communication networks. In this paper, we explore their use in the routing of air traffic control in the arrival space. The algorithms could, in principle, be used in both en-route *Center* airspace or in the *Terminal Radar Approach Control (TRACON)* regions around airports. In this paper, we focus on the terminal area, which is modeled as a (planar) discrete graph that captures the topology of the arrival space area, distances between VOR nodes, entry nodes (TRACON gates), and exit nodes (landing nodes). Initially, all aircraft are assumed to be on the entry nodes, and move with the same, constant velocity throughout. The restrictions of planarity and uniform aircraft velocity are not necessary for the conceptual development of the algorithms. Our goal is to route the aircraft to the exit nodes, which model the beginning of the final approach, without two aircraft being in *conflict*. A conflict occurs whenever two aircraft get closer than 5 miles from each other. Our airport model captures many features of the existing airspace structure. The discrete nature of the model, and the problems we address are similar to the problems discussed in [4].

Optimal conflict-free routing may easily lead to combinatorial optimization problems or dynamic programming problems with undesirable complexity. Solutions to such optimization problems will therefore be unrealistic in practice. Sacrificing some optimality for computational tractability will be crucial for efficient algorithms that guarantee safety while being almost optimal. Greedy hot-potato algorithms reduce the combinatorial complexity by greedily advancing each individual (optical) packet closer to its destination. In the absence of any collision, this will be the optimum solution. However, greedy algorithms may introduce more collisions, especially close to the destinations.

To avoid this situation we consider a variant of the so-called deflection routing algorithms. We begin by pre-computing (off-line) the greedy solution for each node-destination pair of the airport. The can be efficiently done using Dijkstra'a algorithm. As a result we obtain the shortest-paths tree for each destination node which can be used to optimally route aircraft in the absence of collisions. This will be our first guess at routing, thus it is called the *primary routing*. For each node of the airport, we also compute a hierarchy of *deflection nodes* which will be used in case a conflict is detected. This can also be done efficiently off-line, and we can hierarchically rank each deflection edge based on the extra time or distance it adds to the route of the aircraft.

The deflection routing algorithm then uses the primary routing and hierarchy of deflections but in a greedy manner. Aircrafts are initially on the entry nodes and have optimal primary routes to their desired destinations. At each time step, conflicts are predicted for some time in the future. If there are no conflicts then each aircraft proceed along their primary routes. If a conflict is predicted, then one of the involved aircraft must choose a deflection, starting with the one that introduces minimum delay to its overall distance. If that is not enough, then it proceeds to the next available deflection while paying a higher price. The deflection routing algorithm shares similarities with the model predictive control framework, with the main difference that no optimization is performed on-line, as the available solutions (primary routes and deflections) have been pre-computed.

The outline of this paper is as follows: In Section 2 we formulate the problem of obtaining conflict-free routes for multiple aircraft. In Section 3, we review some basic ideas from hot potato routing in optical networks, and in Section 4 we describe the deflection routing algorithm in the context of air traffic management. Section 5 concludes this paper with many topics for further research.

2 Problem Formulation

In this paper, an airport can informally be thought of as a planar graph. The arrival area of an airport (TRACON) consists of a finite number of entry points called the TRACON entry gates. Aircrafts enter the TRACON area via these entry nodes only. Each of these planes need to be directed to a destination node, which model the beginning of the final approach for landing. A plane which reaches the destination node has been given permission to land. We assume that all planes enter and move with uniform and constant velocity. This assumption can be relaxed in future work. Between the entry nodes and the destination nodes there exists a finite number of (VOR) nodes through which planes could be routed. In this paper, we consider a 2D model of the terminal area. A 3D model would be conceptually similar, once more at the price of complexity.

More formally, the airport is modeled as a directed graph $G = (V, E, V_{in}, V_{out})$, where $V \subset \mathbb{R}^2$ are the nodes of the airspace around an airport , $E \subseteq V \times \mathbb{R} \times V$ is a the set of routing edges, $V_{in} \subseteq V$ are the entry nodes, and $V_{out} \subseteq V$ are the destination nodes. The number associated with each edge models the distance between two nodes. We also assume that the graph is acyclic with no holding patterns, and assume that there are no incoming links to the entry nodes, and similarly, there exist no outgoing edges for the destination nodes. Relaxing these assumptions will be considered in the future. A sample airport configuration is shown in Figure 1. It is clear, that even though Figure 1 models the arrival area, the model can also capture en-route traffic.

There are n aircrafts on the network. Each aircraft i has a source node $S_i \in V_{in}$ and a destination $D_i \in V_{out}$. Initially all aircraft are on the source nodes. In Figure 1 there are six entry nodes, six aircrafts, and two destination nodes. The dynamics of the aircraft on the graph is straightforward, each aircraft flows along an edge with constant velocity. Each aircraft i also comes equipped with the *protected zone* P_i, a disk of radius 2.5 miles centered at the center of the aircraft. A *conflict* between aircraft i and j occurs if $P_i \cap P_j \neq \emptyset$.

A *route* or a *path* for aircraft i with source-destination pair (S_i,D_i) is a sequence of edges that start at S_i and end at D_i. That is a path p_i for aircraft i is a sequence $v_0 \xrightarrow{d_0} v_1 \xrightarrow{d_1} v_2 \cdots \xrightarrow{d_n} v_n$ with $v_0 = S_i$, $v_k = D_i$, and for each $0 \leq j \leq k-1$ we have $(v_j, d_j, v_{j+1}) \in E$. The cost of path p_i is simply $J(p_i) = \sum_{j=0}^{n-1} d_j$.

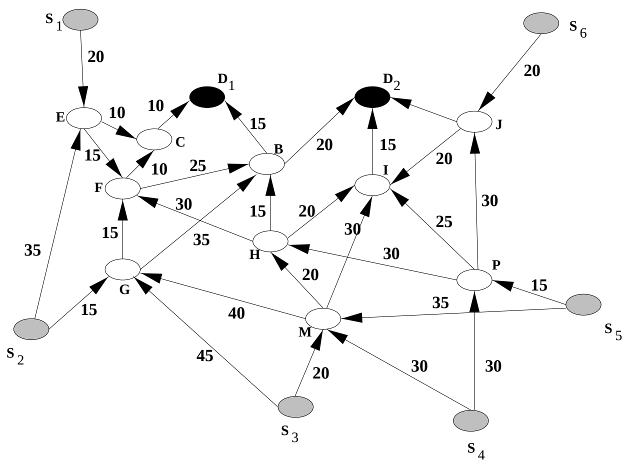

Figure 1: Airspace model around an airport

Our problem formation is now straightforward. Given an airport model $G = (V,E,V_{in},V_{out})$, n aircrafts on the network, and a set of source-destination pairs (S_i,D_i) for each aircraft $(1 \leq i \leq n)$, determine a set of conflict-free paths of minimum cost. Note that in this formulation, aircraft must go to their desired and predetermined destinations. One can also consider the more relaxed problem where aircraft can go to *any* of the destinations.

3 Hot-Potato Routing

Hot-potato routing algorithms [2, 1] use the principle that every packet arriving at a node has to be redirected immediately to any of the its neighboring nodes. This important class of routing algorithms are important in optical networks since nodes of the network are not capable of buffering any packet. Two packets at the same node at the same time are said to have collided if they are directed through the same outgoing link, in which case one of them has to be removed from the network and the other is permitted to proceed. Hot potato algorithms try to route packets by avoiding collisions and optimizing properties such as throughput maximization (deliver the maximum number of packets from source to destination nodes), or minimum global path (the sum of all the distances travelled by the packets in the network). Clearly, similar problems are of great interest in air traffic management systems.

The general problem of routing packets from source to destination pairs without conflict is equivalent to the problem of finding disjoint paths on a graph. However, this problem is known to be NP-hard [9]. One way of relaxing the complexity is to use greedy algorithms to route packets, resulting in *greedy hot-potato deflection routing* algorithms [2]. Our air traffic routing algorithms are inspired from such algorithms.

3.1 Greedy Hot-Potato Deflection Routing

A hot-potato routing algorithm is called *greedy* [2] if each packet (say p_i) at a node is forwarded closer to its destination whenever possible. However if such an assignment of a new routing node leads to a collision with another packet (say p_j) , either packet p_i or the packet p_j needs to be *deflected* to another node that does not necessarily advance it closer towards its destination. The choice of the deflection node can also be done in a greedy sense. Greedy deflection algorithms are particularly attractive because they are simple and have performed well under practical circumstances. These algorithms are also adaptive, that is in low load situations each packet follows the shortest route to their individual destinations. Furthermore, these algorithms are well suited in dynamic scenarios where packets are injected dynamically into the network.

4 Hot Potatoes in the Sky

In this section, we describe a deflection algorithm for routing air traffic. The algorithm consists of three distinct parts.

1. **Compute Primary Routes (Off-line)**
2. **Compute Deflection Routes (Off-line)**
3. **Greedy Deflections (On-line)**

Intuitively, for each destination node of the network, primary routing computes optimal routes for that particular destination from any other node of the network (in the absence of any collisions). This can be efficient performed off-line and stored in the form of a tree. In the presence of conflicts, however, planes must be deflected. Given the output of primary routing, for each node we compute a variety of deflection edges (routes), which are ranked based on how non-optimal the deflection is. This can also be efficiently performed off-line, and stored for each node. Therefore, given a desired destination, we have for each node a primary (optimal) route, and an array of local deflections in case of conflicts.

What remains to be done is online is very simple. Aircrafts begin at the source nodes. If there are no conflicts *predicted* for the next T minutes (typically 10-20 minutes), then the aircraft proceed along their primary routes. Efficient conflict prediction is critical for this approach, and has been sufficiently addressed in the literature. If a conflict is predicted between two aircrafts, then one of the aircraft will take the earliest available deflection. In case there are many deflections to choose from, then the aircraft will choose greedily, that is it will choose the deflection which results in the smallest price. Clearly, the earlier the prediction, the more deflection choices to choose from, the more optimal the re-routing. We now describe each aspect of the overall algorithm.

4.1 Primary Routing

Primary routing is concerned with computing optimal paths from any airport node to each of the destinations in the absence of any network traffic. Ideally, under light load conditions, traffic should be routed using the primary routes.

Recall that we are given a weighted, directed graph $G = (V, E, V_{in}, V_{out})$, with a weight function $w : E \to \mathbb{R}$ which assigns to each edge $(v_0, d_0, v_1) \in E$, the natural number d_0, which is the length of the edge. The weight of a path $p = v_0 \overset{d_0}{\to} v_1 \overset{d_1}{\to} v_2 \cdots \overset{d_n}{\to} v_n$ is simply the cost

$$w(p) = \sum_{j=0}^{n-1} d_j$$

Given two nodes v, $v' \in V$, let $p(v, v')$ be the set of all paths starting at v and ending at v'. Then we define the shortest-path from v to v' as

$$\delta(v, v') = \min_{r \in p(v,v')} w(r) \qquad (1)$$

If there is no path from v to v' then $\delta(v, v') = \infty$. In a shortest-path problem, we wish to compute not only the shortest-paths between all source-destination pairs, but also the vertices along the shortest-paths as well. To be more precise, we will be interested in computing the *shortest-paths tree* rooted at each destination node $d \in V_{out}$.

A shortest-paths tree for a weighted, directed graph G=(V,E) rooted at $d \in V$ is a directed subgraph $G' = (V', E')$ of G, which is a tree, where $V^{'} \subseteq V$ and $E^{'} \subseteq E$, such that

1. $V^{'}$ is the set of vertices reachable from d in G,
2. $G^{'}$ forms a rooted tree with root d, and
3. for all $v \in V^{'}$, the unique simple path from d to v in the tree $G^{'}$ is a shortest (but not necessarily unique) path from d to v in G.

We should note that the shortest-paths which are unique in the tree G', are not necessarily unique in the original graph G, and neither is the shortest-paths tree G'.

Dijkstra's algorithm is an efficient algorithm for computing a shortest-paths tree rooted at a source node in $O(|V|^2)$ running time, where $|V|$ is the number of nodes. This algorithm is used to compute the single-source, shortest-path problem, i.e the shortest path from a source node d to any other vertex. In our case, rather than the single source problem, we are interested in computing the single-destination shortest-path problem for each of the destination nodes. This can be achieved by reversing the directions of the edges in the graph.

If we run Dijkstra's algorithm on a directed weighted directed graph with non-negative edge weights and a source node, then at the termination we will have the shortest-paths tree rooted at node d with $D(v) = \delta(v, d)$ computed for all vertices $v \in V$, where $D(v)$ stands for the *depth* of node v in the tree G'. We should note this depth represents the magnitude of the shortest-path to the destination d, and the not the usual depth of a tree.

Applying Dijkstra's algorithm on our airport graph G with the edge directions reversed for each destination node $d_i \in V_{out}$, we can obtain the shortest-paths tree associated with the destination d_i. For each destination $d_i \in V_{out}$, we will denote each such tree rooted at a destination d_i with SPT_{d_i}. Consider any tree SPT_{d_i} that we obtain by this procedure; at each node there may exist many incoming edges, but only one outgoing edge. Thus at every node, the next node the aircraft must visit on it's way to destination d_i is uniquely determined. Therefore, given any node on the network, $SPT(d_i)$ gives a unique (in the tree G'), optimal route to destination d_i. This destination dependent route will be called the primary route. Furthermore, for each node $v \in V$ and each destination $d_i \in V_{out}$, the tree $SPT(d_i)$ uniquely determined the primary (next) routing node at v.

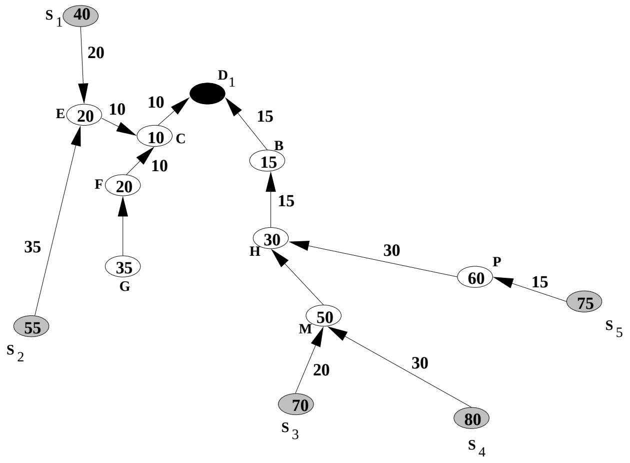

Figure 2: SPT(D_1) for destination D_1

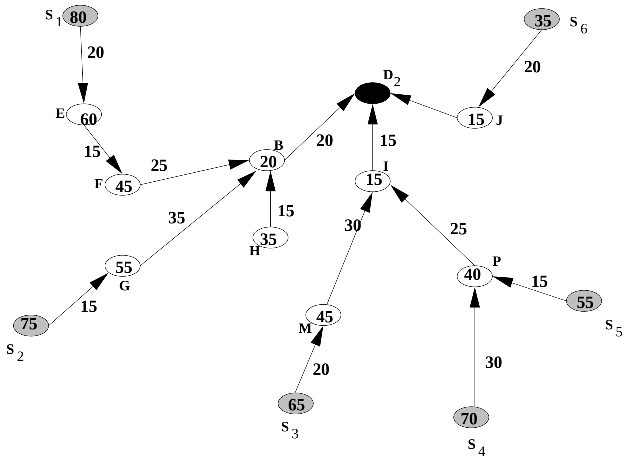

Figure 3: SPT(D_2) for destination D_2

Reconsider the airport model shown in Figure 1. Using Dijkstra's algorithm we obtain two SPT's; one for each destination. The tree $SPT(D_1)$ is shown in Figure 2, and $SPT(D_2)$ is shown in Figure 3. The number inside each node is the depth of that node, which is simply the optimal distance from that node to the desired destination. Considering node F, we see that for destination D_1, the primary routing node is C, however for destination D_2 it is node B. We should note that nodes from which we cannot reach a destination do not exist in the corresponding trees.

4.2 Deflection Routes

Primary routes will work very well in the absence of conflicts, and will be the first choice of incoming aircraft given desired destinations. In a loaded network, planes need to be re-routed or deflected. In this section, we describe how to pre-compute all the deflection nodes for each node that are relevant to a given destination. The deflection routes will use the information encoded in the shortest-path tree SPT_{d_i} computed above.

To compute possible deflection routes at a particular node while still reaching the same destination d_i, we run through each node of $SPT(d_i)$, and inspect each corresponding outgoing edge in the original graph G (excluding the primary node in $SPT(d_i)$). Consider for example node P with desired destination D_2. In Figure 4, which shows $SPT(D_2)$, the primary routing node for destination D_2 is node I. In case an aircraft at node P needs to be deflected (while heading to destination D_2), then in the original graph G of Figure 1 there are two non-primary edges. One edge heads to node H and another one to node J. These two deflection edges are shown in Figure 4 using dashed arrows.

The price to pay for taking any of these two deflections can be easily quantified. The optimal distance from P to D_2 is 40. Since the optimal distance from H to D_2 is 35 (assuming the primary is taking from H) and the edge from P to H has length 30 in G, then the extra price to pay is 25. Note that there is no guarantee that the primary will be taking from H onwards, as more conflicts may arise later. Similarly, the extra price to pay if the plane is deflected to node J can be easily computed to be equal to 5.

The procedure we just described for node P can be easily performed for each node in V, and for each desired destination $d_i \in V_{out}$. As a result, for each desired destination we can pre-compute at each node a set of deflection routes or nodes, along with the *deflection penalty* associated for choosing that deflection. One can intuitively think of the deflection penalty as the difference in the primary routing cost of the two nodes of the deflection edge. We therefore sort the deflection edges at each node based on their deflection penalty.

We should note that contrary to the primary routes which have been obtained while globally minimizing the distance from each node to the desired destinations, the deflection penalty is a local quantity and only provides local information. Therefore, deflections may be non-optimal, but on the other hand, they quite easy to compute. In the next section, we shall use these computations in order to avoid potential conflicts while minimizing the deflection penalty.

4.3 Greedy Deflections

The online algorithm consists of routing aircrafts to there respective destinations conflict free. Our off-line computations are going to assist us to determine such routes and also how to react in case we predict

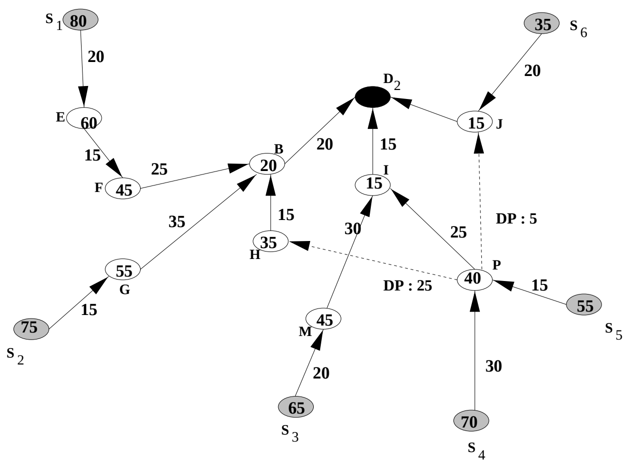

Figure 4: Deflection nodes at node P for destination D_2

a collision sometime in the near future. To determine whether any two aircrafts are conflict-free, we predict their paths along their primary routes for T minutes ahead (typically $T = 15$ or 20 minutes). It is a straight-forward computation to determine whether these paths are in conflict [12]. Larger values of T we result in more strategic conflict resolution, but at the price of higher complexity. Small values of T will make our routing decisions very myopic. In case we predict that the primary paths of two aircrafts will lead to a conflict in the next T minutes, it would be necessary to deflect one of the aircrafts via another node.

In order to make deflections as strategic as possible, we choose the *earliest possible deflection* that is available for either aircraft. If the choice is not unique, then we could be confronted with a number of deflection choices for the same destination. We then greedily choose a route with the least deflection penalty.

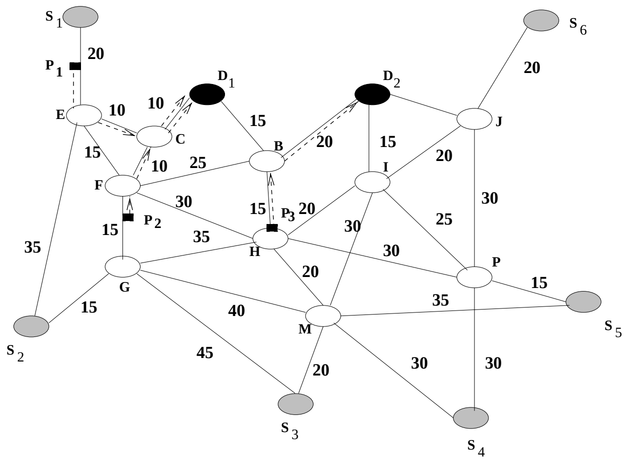

Figure 5: Primary routes in conflict

For example let us consider the scenario of three planes P_1 located 10 miles from node E destined to D_1, P_2 located 5 miles from node F destined to D_1, and P_3 right on node H destined to D_2. Let us assume that all the aircraft are moving at 160 miles per hour and we are predicting collisions 15 minutes ahead. This is equivalent to looking ahead 40 miles for each planes. We first try to route each plane via there primary routes (obtained from their corresponding $SPT(D_i)$'s. The primary routes 40 miles ahead are shown as dashed lines in Figure 5. Cleary such a routing leads to collision between aircrafts P_1 and P_2. However there are no collisions predicted with the path of P_3. Now to avoid collision between P_1 and P_2, we need to deflect P_2 at node F and route it to node B. However such a routing leads to a new collision prediction (40 miles ahead) previously nonexistant between P_2 and P_3. This creates the so-called domino effect. One way of resolving the domino effect is to place a hierarchy in the conflict resolution algorithm. For example, aircraft closer to their destination get routed first. In this example, P_1 will have a higher priority then P_2, which has a higher priority than P_3. So we now deflect P_3 via F to D_2. Now these set of obtained paths are collision free, as shown in Figure 6.

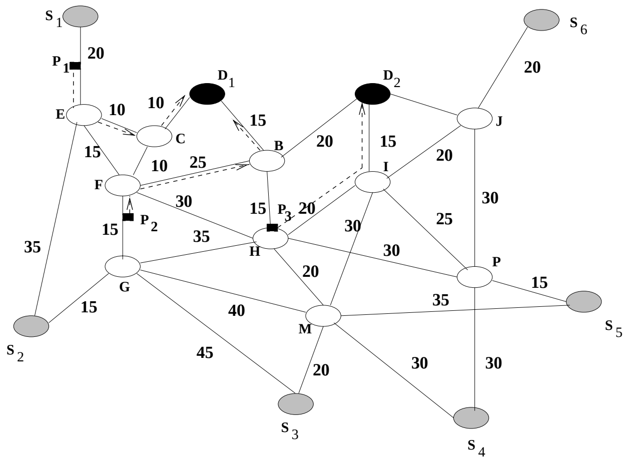

Figure 6: Conflict-free routes after deflections

5 Conclusions

In this paper, we have explored the use of hot potato routing algorithms from optical communications to the problem of air traffic management. It is our hope that this line of research will establish stronger connections between the air traffic management and optical communications community.

Future work will focus on formally analyzing the complexity of the algorithms, introduce holding patterns, and 3D airspace, relax the uniform velocity assumption, dynamic re-route aircraft in dynamic network traffic, and route aircraft to any of the available destinations.

Acknowledgment: This research is partially supported by the University of Pennsylvania Research Foundation.

References

[1] A. Ben-Dor, S. Halevi, and A. Schuster. Potential function analysis of greedy hot-potato routing. *Theory of Computer Systems*, 31(1):41–61, January 1998.

[2] A. Borodin, Y. Rabani, and B. Schiber. Deterministic many – to – many hot–potato routing. *IEEE Transactions on Parallel and Distributed Systems*, 8(6):587 – 596, 1997.

[3] Christopher R. Brinton. An implicit enumeration algorithm for arrival aircraft scheduling. In *Proceedings of the 11th Digital Avionics Systems Conference*, Seattle, WA, October 1992.

[4] S. Devasia and G. Meyer. Automated conflict resolution procedures for air traffic management. In *Proceedings of the IEEE Conference on Decision and Control*, pages 2456–2462, 1999.

[5] Heinz Erzberger, Thomas J. Davis, and Steven Green. Design of center-tracon automation system. In *Proceedings of the AGARD Guidance and Control Syposium on Machine Intelligence in Air Traffic Management*, pages 11.1–11.12, 1993.

[6] W. H. Harman. TCAS : A system for preventing midair collisions. *The Lincoln Laboratory Journal*, 2(3):437–457, 1989.

[7] D.R. Isaacson, T.J. Davis, and J. E. Robinson III. Knowledge-based runway assignment for arrival aircraft in the terminal area. In *Proceedings of the AIAA Guidance, Navigation and Control Conference*, New Orleans, LA, August 1997.

[8] Stephen Kahne and Igor Frolow. Air traffic management: Evolution with technology. *IEEECS*, 16(4):12–21, 1996.

[9] Joseph Naor, Ariel Orda, and Raphael Rom. Scheduled hot-potato routing. *Journal of Graph Algorithms and Applications*, 2(4):1–20, 1998.

[10] Russell A. Paielli and Heinz Erzberger. Conflict probability estimation and resolution for free flight. NASA Ames Research Center, Preprint, 1996.

[11] Nicolas Pujet and Eric Feron. Flight plan optimization in flexible air traffic environments. Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, 1995.

[12] Claire Tomlin, George J. Pappas, and Shankar Sastry. Conflict resolution for air traffic management : A study in muti-agent hybrid systems. *IEEE Transactions on Automatic Control*, 43(4):509–521, April 1998.