NETWORK ROUTING MODELS APPLIED TO AIRCRAFT ROUTING PROBLEMS

Zhiqiang Chen, Andrew T. Holle,
Bernard M.E. Moret, and Jared Saia

Department of Computer Science
University of New Mexico
Albuquerque, NM 87131, U.S.A.

Ali Boroujerdi

Naval Research Laboratory
4555 Overlook Avenue SW
Washington, DC 20375, U.S.A.

ABSTRACT

We study network models applied to two aircraft routing problems, one in which the goal is to route strike aircraft to a target and back so as to minimize losses and one in which the goal is to route civilian traffic around an airport so as to minimize noise exposure to the population. We propose *joint routing* as our model: find a required number of time-disjoint routes in a network that minimizes the total cost. We show that joint routing can in turn be modelled as a dynamic network flow problem. We study this problem under several variants and on different types of networks, establishing tight bounds on the running time of exact solutions through applications of both existing and some new methods. We also discuss the modelling of the airspace in which the routing takes place and how choices affect the performance of our optimization algorithms. Our model extends to other applications, such as the routing of hazardous materials and of secure communications.

1 INTRODUCTION

Aircraft routing requires the generation of non-colliding, time-dependent routes through a specified airspace. Constraints may include forbidden regions, required waypoints, flight corridors (in civilian aviation), and time windows through various regions. We study two variants of the problem: (i) planning strike missions to destroy selected targets in the presence of enemy threats, where the objective is to maximize payoff; and (ii) routing civilian aircraft around an airport, where the objective is to minimize the noise exposure of the local population. Our model, which we call *joint routing*, extends to other applications, including some where the objects routed are not aircraft and may in fact stop en route, such as the design of transport routes for hazardous materials and that of secure communications within a network.

In joint routing, we model the airspace as a network and seek a collection of routes through the network with the following characteristics: (i) the aircraft are in constant motion; (ii) the aircraft do not collide; (iii) the aircraft all reach their destination; and (iv) the collection of routes optimizes the chosen objective. We show that joint routing can be modelled as flow within a space-time network, an application of the dynamic network flow technique of Ford and Fulkerson (1958). Since dynamic network flow is very expensive, we present a careful analysis of the running time of our algorithms, including a discussion of the effect of network design on their performance.

A number of related problems appear in the operations research literature, notably the vehicle routing and scheduling problems and other transportation problems: surveys include Bielli *et al.* (1982) and Solomon (1988), while a recent issue of *Operations Research* (1993) was devoted to the problem of aircraft scheduling and routing. Of all these models, however, only one closely resembles the aircraft routing problems just described: the dynamic traffic assignment problem as modelled through flow in a space-time network (Zawack and Thompson 1987). However, the traffic assignment problem is constrained by a fixed time period, while joint routing has no such constraint; moreover, as in other references to dynamic network flow techniques, Zawack and Thompson omit a formal analysis of performance.

In the following, we briefly review the dynamic network flow model of Ford and Fulkerson, discuss its application to joint routing, and analyze the running time of various approaches. In particular, we prove that, if there exists a feasible set of routes in a network, then there exists a set of routes where each route crosses at most $O(|V|)$ edges, even on networks with cycles; this result allows us to bound the number of time steps that must be considered in the joint routing problem and thus allows us to conclude that joint routing can be solved in polynomial time. We then address the issue of network design in modelling airspace and its influence on the performance of our algorithms, presenting evidence that careful design can improve running times by large factors, both in asymptotic terms and in practical applications. An early version of this work appeared in Boroujerdi *et al.* (1993).

2 THE PROBLEM

A formal statement of our problem (in its basic version) can be written as follows.

- Given a graph with two distinguished vertices, a sink and a source, and where each edge has an associated cost, route aircraft from the source to the sink, subject to the constraints:

  1. aircraft are initially placed at the source;
  2. at each step, every aircraft crosses an edge of the graph (no aircraft can stop at a vertex for one or more steps);
  3. both edges and vertices have unit capacities (so that aircraft cannot collide at vertices or along edges).

  The goal is to find a least-cost routing for a given number of aircraft.

A key assumption underlying the problem is that aircraft and parameters are uniform. In the absence of such uniformity, the problem is generally $\mathcal{NP}$-hard.

Theorem 1. The joint routing problem is $\mathcal{NP}$-hard under any of the following conditions:

- objects move at different speeds and incur different costs
- certain edges may not be crossed by certain objects
- the cost of crossing an edge depends on the object crossing it
- the cost of crossing an edge depends on the time step
- the total cost is the sum of the costs of edges that are crossed at least once

The first four of these $\mathcal{NP}$-hardness results are new, but not unexpected (we omit the difficult proofs, since the constructions are not of special interest; the interested reader should consult Boroujerdi 1994); the fifth is known—it is the "minimum edge-cost flow" problem mentioned in Garey and Johnson (1979, p. 215).

3 DYNAMIC NETWORK FLOW

In their 1958 paper, Ford and Fulkerson showed how to transform the *maximum dynamic flow problem* to a conventional network flow problem. In a dynamic network flow problem, arcs have traversal times as well as capacities and flow received at a node can either be moved immediately or held over for later discharge; the question is to find the maximum flow that can be moved from the source to the sink within some given period of time T. The reduction uses the given network, $G = (V, E)$, to build a new network, $G^* = (V^*, E^*)$, as follows. For each node, $v \in V$, create T copies: $v_1, \ldots, v_T$ (a slight correction to the presentation of Ford and Fulkerson, who unnecessarily created $t+1$ copies); in addition, set up a supersource, s^*, and a supersink, t^*. Let $t(v, w)$ be the traversal time of arc (v, w); introduce an arc from v_i to w_j (of capacity equal to that of arc (v, w)) if and only if $j - i = t(v, w)$; in addition, introduce an arc from v_i to v_{i+1} with infinite capacity for each $v \in V$ and $0 \leq i < T$; finally, connect the supersource to every copy of the original source and connect every copy of the original sink to the supersink, all with arcs of unbounded capacity. Then the maximum amount of flow that can be moved from the source to the sink in T time units in G equals the maximum flow from s^* to t^* in G^*. If the flow received at a node, v, must leave it immediately, the reduction omits the arcs from v_i to v_{i+1} for all $v \in V$ and all i and omits the supersource—the source for the new network is simply s_0. Note that these reductions create a new graph that is T times larger than the original—although the new graph can be reduced somewhat by eliminating in linear time all vertices and edges that are not on a path from the supersource to the supersink.

Dynamic network flow has since been used by many researchers in modelling various vehicle routing and scheduling problems, particularly in aircraft problems (see, e.g., Bielli *et al.* 1982, Orlin 1983, *Operations Research* 1993).

We observe that dynamic network flow can be used to solve the basic version of our joint routing problem, where costs are absent and the objective is simply to route as many aircraft as possible through the network.

Theorem 2. Routing as many aircraft as possible from the source to the sink in T time units while avoiding collisions is equivalent to solving a dynamic network flow problem (in which flow cannot be stored at nodes) on a network with vertices of unit capacity.

Our earlier discussion establishes the correctness of this theorem; we are not aware of any previous statement of it, although uses of dynamic network flow in vehicle routing and aircraft scheduling offer a close parallel.

4 THE SPACE-TIME NETWORK

The graph generated by the reduction described in the previous section has a very constrained structure; in particular, when flow must leave each node immediately, the result is a directed acyclic graph where each node other than the supersink is at a well-defined distance from the source s_0 (i.e., any path from s_0 to the node traverses the same number of arcs). Ignoring the supersink, we call such a graph a *time-transformed acyclic graphs* or *trag.* Note that several different graphs can give rise to the same trag; we define the *order* of a trag to be the size (number of vertices) of the smallest graph that can generate the trag. The correspondence between a trag and a graph that generates it helps define the structure of trags, which in turn enables us to characterize the behavior of network flow algorithms on trags.

We define the *rank* of a trag vertex as the distance from s_0 to that vertex; each vertex has a well-defined rank (since the supersink is excluded from the trag). We observe that distinct trag vertices of the same rank must correspond to distinct vertices in the graph that gave rise to the trag; in particular, since the original graph has a single sink, the trag can have at most one sink vertex at each rank. We define a *subtrag* of some given trag T as the subgraph of T induced by a vertex of T and all of the vertices of T reachable from that vertex; we then generalize this concept to a subgraph of T generated by a subset of vertices of T of the same rank such that the induced graph cannot be decomposed into disjoint pieces. If two subtrags of some trag T are isomorphic, they may simply be copies of the same structure within the original graph (which generated T); however, our observation about rank shows that two isomorphic copies of a subtrag, whose generating vertices have the same rank, must be generated by distinct structures in the original graph. This property allows us to bound the size of the generating graph for a given trag T.

Let T be a trag; define S_i to be the set of maximal subtrags of T induced by (subsets of) vertices of rank i. (By maximal, we mean that no subtrag in S_i can properly contain an isomorphic copy of another subtrag in S_i.) Note that the subtrags of S_i must be pairwise disjoint. We claim that the order of T cannot exceed the sum over all ranks of the number of sources in each of the S_is; in fact, we can build a directed graph G that will generate T when replicated: for each subtrag in S_i, place its sources within G, then connect a vertex u of G to another vertex v of G whenever u at some rank j is connected to v at rank $j+1$ in T. Now let $\mathcal{T}$ be the set of subtrags of T with single source (induced by a single node) partitioned into equivalence classes under isomorphism (each class thus contains a collection of isomorphic subtrags); for equivalence class c, define $r(c)$ to be the maximum over all ranks i of the number of isomorphic subtrags in c with source of rank i. We claim that the order of T must be at least as large as the sum of the $r(c)$ over all equivalence classes $c \in \mathcal{T}$: this is a direct consequence of our earlier remark that two isomorphic copies of a subtrag with sources of the same rank must be generated by distinct structures in the original graph.

These and other properties of trags currently under study have allowed us to derive tighter bounds on the best possible routings of aircraft and to devise heuristics to speed up the running time of optimization algorithms. In particular, they have allowed us to characterize graphs that give rise to series-parallel and nearly series-parallel trags—an important characterization as network flow is solvable in linear time on series-parallel graphs.

5 RESULTS ON JOINT ROUTING WITHOUT COSTS

In order to establish a bound on the running time of a solution algorithm, we need to bound the number of time steps, T, that must be used in the reduction to dynamic network flow.

Theorem 3. Routing a maximum number of objects from source to sink without collisions can be solved in polynomial time.

Proof: We have seen that we can answer the question by solving a dynamic network flow problem; hence we need only place a bound on T that remains polynomial in the size of the network. If the network is a directed acyclic graph, there is no problem: no object can take longer than $n-1$ steps (where n is the number of vertices in the graph) before reaching the sink. If the network allows cycles (either as an undirected or as a directed graph), a tight bound is more difficult to derive (see below), but a safe upper bound of n^2 steps is easily established as follows.

Consider the state of the system during the first n steps; if, at the end of n steps, one of the objects has reached the sink, then the problem reduces to routing $n-1$ objects through the network and the conclusion follows by strong induction. If no object has reached the sink by step n, then there must exists some index $i < n$ at which the object closest to the sink (in terms of the number of edges along a shortest path to the sink) is $n-i$ edges away from it. Pick the largest such index: the object closest to the sink can now safely be moved directly to the sink along the shortest path, since no other object can collide with it along that path; the other objects are moved as in the original sequence. The new sequence now sends at least one object to the sink in n steps without collision, reducing the problem to the first case.

When time windows are added (say a minimum time of T_0 and a maximum time of T_1), they can be included in the dynamic network by the simple expedient of (i) duplicating the network only for the first T_1 steps; and (ii) removing all arcs leading into the sink from vertices with associated times smaller than T_0. Note that deadlines do not affect the proof of the theorem—nor, indeed, any of the results below.

In fact, the number of edges crossed by any of the objects on its way from source to sink remains linear in the size of the graph, rather than quadratic. Proving such a linear bound is relatively simple for undirected graphs, but much more challenging for directed ones; we give only the former proof.

Theorem 4. Given an undirected graph with two distinguished vertices (a source and a sink), if k objects can be routed from the source to the sink without collision, then they can be so routed in $O(n)$ steps, where n is the number of vertices of the graph.

Proof: We prove that, in an undirected graph, object $i+1$ need never arrive more than 2 units of time after object i; since the first object to arrive can always arrive in at most n steps, this establishes a bound of $n+2(n-1)$ steps on the arrival time of the last object.

If there is a feasible joint routing schedule for the n objects, then there is a feasible schedule which minimizes the sum of all arrival times; in this schedule, we index each object by its order of arrival to the sink—we shall use induction on this index. Consider then the ith object and look at the last cycle it traces on its path from source to sink. This cycle is present to avoid a collision with some object j at some vertex v; from v to the sink, object i has a path without cycle. We claim that we must have $j<i$; if the cycle were to avoid collision with a slower object ($j>i$), we could reduce the sum of all arrival times by letting object j use object i's path from v to the sink and thus arrive faster with no collision. Now, observe that the last cycle traced by object i has length 2; otherwise, we could reduce the sum of all arrival times by shortening the cycle to a length of 2 without creating new collisions for object i (rather than going around the cycle, object i would simply traverse the first edge of the cycle forward and backward). Hence object i arrives at vertex v 2 units of time after object j and thus arrives no more than 2 units of time after object j at the sink. Since $j<i$, object j obeys the induction hypothesis and our conclusion follows.

While these results indicate that the basic version of joint routing without edge costs is solvable in strongly polynomial time, the solution algorithm appears inefficient, since it is a maximum flow algorithm running on a graph expanded by a factor of $|V|$; as most maximum flow algorithms scale according to $|V|^3$, the expansion appears to cost us a factor of $|V|^3$ in running time. However, the networks used in our model have unit vertex capacities, so that the expansion creates, in effect, a unit network (i.e., a network in which all arcs have unit capacity, at most one of the arcs (v,w) and (w,v) is present, and every vertex has either in- or out-degree of one). The maximum flow in a unit network of n vertices and m edges can be found in $O(\sqrt{n}\cdot m)$ time by using blocking flows. Using a linear bound on T, the expansion of a network, $G=(V,E)$, results in a unit network, $G^*=(V^*,E^*)$, with $\Theta(|V|^2)$ vertices and $\Theta(|E|\cdot|V|)$ edges. Thus the maximum flow in G^* can be found in $O(\sqrt{|V^*|}\cdot|E^*|)=O(|V|^2\cdot|E|)$ time. Note that an algorithm based on augmentation along single augmenting paths also runs in $O(|V|^2\cdot|E|)$ time. To see how, we note that the value of a maximum flow in G^* and hence the number of augmentations cannot exceed $|V|$. An augmenting path can be found in linear time by performing a depth- or breadth-first search in a graph with $O(|V|\cdot|E|)$ edges, resulting in a total running time of $O(|V|^2\cdot|E|)$. (Preflow-based methods, on the other hand, while the best on general networks, seem unable to make use of the special structure of unit networks.)

Finally, in a sparse network, we expect that only a few of the n duplicates (of vertices or edges) will be used. In order to reduce storage costs, we can maintain a list of used time slots for each vertex and for each edge. The list manipulations add some expense to the running time, but gain considerable space and, on average, still expend very little time per edge or vertex in identifying the correct slot.

6 RESULTS ON JOINT ROUTING WITH COSTS

The min-cost max-flow problem long resisted strongly polynomial solutions. Tardos (1985) first provided such an algorithm for minimum-cost circulations and thus also for min-cost max-flow problems. However, in the case of networks with polynomially bounded maximum flow, min-cost max-flow problems are easily solved in (strongly) polynomial time by the simplest of all min-cost flow algorithms: at each step, augment along a path of minimum cost. At each step, the flow must increase by at least one unit; since the max-flow cannot exceed some polynomial bound B (in our case, $B = |V|$), the number of augmenting steps is polynomial. Each step requires solving a shortest-path problem, which can be done with Dijkstra's algorithm (implemented with Fibonacci or relaxed heaps) in $O(|E| + |V| \log |V|)$ time. The complete algorithm thus runs in $O(B \cdot (|E| + |V| \log |V|))$ time. The expanded graph, G^*, is constructed as in Theorem 2, except that a cost is associated with each arc of G^*—that of the corresponding edge of G. The costs of the (t_i, t^*) arcs can be anything as long as they are all equal. Then the minimum-cost maximum set of routes in G is equal to the minimum-cost maximum flow in G^*.

Theorem 5. Given a graph, $G = (V, E)$, with a source, s, and a sink, t, and given a cost function, $c: E \rightarrow \mathcal{R}_0^+$, denoting the cost of crossing the edges of G, routing the maximum number of objects from s to t at minimum cost is solvable in strongly polynomial time.

Proof: First, we prove that the number of time steps, T, required to route the maximum number of objects from s to t at minimum cost is polynomial in the size of G. Let R be a minimum-cost set of routes sending the maximum number of objects from s to t. If each route in R is a minimum-cost path from s to t, then T is trivially bounded by $|V|$. Otherwise, consider a minimum-cost path, p, of maximum length from s to t, $(s = v_{\pi(0)}, v_{\pi(1)}, \ldots, v_{\pi(k-1)}, v_{\pi(k)} = t)$. There must exist an index $0 < i < k$ such that $v_{\pi(i)}$ is occupied at time i, or p can be used in place of a more costly route, resulting in a set of routes less costly than R, a contradiction. Pick the largest such index. Since p is the longest minimum-cost path from s to t, the object occupying $v_{\pi(i)}$ at time i must reach t in at most another $(k - i)$ steps. Thus, after $k < |V|$ time steps, at least one object reaches the sink. Now let p' be a minimum-cost path to t of maximum length from any vertex occupied at time k. By a similar argument, at least one more object must reach the sink in another $k' < |V|$ time steps, where k' is the length of p'. Since there are now fewer than $|V|$ objects to route, applying the argument inductively proves that the last object to reach the sink cannot take more than $|V|^2$ time steps.

An analog of Theorem 4 also holds.

Thus, to compute in strongly polynomial time a minimum-cost set of routes in a graph, G, we need only compute a minimum-cost maximum flow in the corresponding expanded graph, G^*, by successive augmentations along minimum-cost augmenting paths. Using linear bounds on the number of time steps, each augmentation requires running a general shortest path algorithm (because edge costs in the residual graph of G^* can be negative) on a graph of $\Theta(|V|^2)$ vertices and $\Theta(|V| \cdot |E|)$ edges, thereby taking $O(|V|^3 \cdot |E|)$ time. Since the value of a maximum flow and hence the number of augmentations cannot exceed $|V|$, the running time of the algorithm is $O(|V|^4 \cdot |E|)$. More sophisticated minimum-cost maximum-flow algorithms that perform better in general do not yield any better bounds for this problem.

As noted earlier, adding multiple sources and multiple sinks is easily handled. Time windows are implemented very naturally: if objects can only pass through some vertex v during time window $[i \ldots j]$ and assuming that j is less than $|V|$, we only place in the expanded network vertices $v_i, \ldots, v_j$, omitting vertices $v_0, \ldots, v_{i-1}$ and $v_{j+1}, \ldots, v_{|V|}$. Should a time window at the sink delay arrival to the sink for a long period ($i \gg |V|$), then we shall be forced to expand the network to $T = j$ and thus cause much larger execution times, possibly only pseudo-polynomial (depending on T rather than on $\log T$ or just $|V|$); however, the solution itself will then consist of a set of very long walks in the network, so that no algorithm could run in strongly polynomial time and yet print all routes. Small capacities larger than 1 do not alter any of the results and algorithms; however, arbitrarily large capacities cause much the same problem as arbitrarily large time windows: the network may then support the routing of a correspondingly large number of objects, so that the number of routes to be printed may grow as a function of $c_{\max}$ rather than as a polynomial function of the input. Again, no algorithm could run in strongly polynomial time under these conditions while printing each route. Finally, we note that the extension to object-dependent costs remains tractable if the dependency on the object consists of a simple cost multiplier (a realistic situation): in that case, we compute routes as if all objects had a multiplier of 1 by using the minimum-cost model discussed above, then assign objects to routes by assigning the object with largest multiplier to the least costly route, proceeding greedily down to the final assignment of the object with the smallest multiplier to the most costly route—it is easily seen that this assignment of objects to routes is optimal and that the resulting routing minimizes the overall cost.

7 NETWORK DESIGN

Since the running time of our method depends heavily on the size of the network used in the model, it pays to consider how the network should be generated. In both the military and the civilian aviation applications, the network needs to: (i) model 3-dimensional airspace; (ii) provide sufficient resolution to make optimization meaningful; (iii) describe realistic scenarios (e.g., enable flight paths that remain within allowed operational parameters for the application); and (iv) incorporate as much local information as possible. Of particular importance in modeling the civilian aviation problem, for instance, is the fact that, regardless of how the network is designed, many paths through it cannot be followed by the aircraft, as they include sections with excessive acceleration or unacceptable aircraft positions. In fact, this constraint on acceptable paths turns our joint routing problem into an $\mathcal{NP}$-hard optimization problem again, even in the absence of other parameters.

We designed a variable-resolution grid, with resolution dictated by the terrain, the proximity to critical areas (airport, target, enemy defense, etc.), and the desired final resolution (or, equivalently, the allowable computation time). The grid is set up initially at some fixed low resolution, then recursively refined over the entire airspace according to the parameters just listed. The top of the grid is a flat surface at an altitude considered sufficient to include all possibilities; the bottom of the grid hugs the terrain, while ensuring that no edge ever goes through solid matter. The basic grid (and its recursive refinement) uses a shape adapted to the problem at hand: for the military application, we use a simple 3-dimensional rectangular mesh (distorted as needed by the terrain into trapezoidally shaped cells), while, for the civilian application, we use a 3-dimensional cylindrical mesh (again with cells distorted as needed by the terrain) with edges aligned with the runways in the immediate vicinity of the airport.

Since the constraints on allowable civilian aircraft moves make the problem intractable, we approximate the problem by running our regular augmentation algorithm. This algorithm may now fail to return an optimal solution, since, as it computes a least-cost path to the sink, it may select some path over another only to discover later that the selected path violates the move constraints—too late to recover the second-best candidate. In our experiments to date, the loss of accuracy in optimization remained very small.

In order to speed up the optimization algorithm, we are also looking into special networks. For instance, series-parallel graphs (a special type of network) allow a linear-time solution for the network flow problem; while our copying procedure typically cannot produce series-parallel networks, it also typically produces networks with closely related structure, on which several simple heuristics can greatly decrease the running time of our algorithms. The series and parallel compositions of the series-parallel graphs can be identified within the trag produced by our transformation and the problem accordingly decomposed into a collection of smaller subproblems, with large resulting gains in efficiency.

8 FURTHER WORK

We have implemented several versions of these algorithms for joint routing both with and without costs. In all cases, the simplest methods are the fastest on our (to date somewhat limited) suite of test cases; in particular, the scaling circulation methods proposed for the min-cost problem are slow when compared to the simple method of augmenting along a path of minimum cost—the more sophisticated methods spend too much time in establishing labellings at each iteration. In the absence of other known methods for computing these routes and of known optimal solutions (in many cases, the objective function remains incompletely defined, due either to somewhat unclear goals or to inaccurate cost functions), it is difficult at this stage to quantify the quality of the solutions obtained. In a next stage, we shall generate routes for actual military planning and civilian routing problems, which will allow us to compare our solutions with those obtained by existing methods. Preliminary data suggest significant improvements.

ACKNOWLEDGMENTS

This work is supported by the Office of Naval Research under contract N00014-92-C-2144.

REFERENCES

Ahuja, R. K., T. L. Magnanti, and J. B. Orlin. 1993. *Network Flows.* Prentice-Hall, Englewood Cliffs, N.J.

Bielli, M., G. Calicchio, B. Nicoletti, and S. Ricciardelli. 1982. The air traffic flow control problem as an application of network theory. *Comput. and Operations Research* 9(4):265–278.

Boroujerdi, A. 1994. *Joint Routing and Persistent Data Structures.* Ph.D. Dissertation, Department of Computer Science, University of New Mexico.

Boroujerdi, A., C. Dong, Q. Ma, and B. M. E. Moret. 1993. Joint routing in networks. Proc. NET-

FLOW 93, San Miniato (Università di Pisa TR-21/93), 175–180.

Ford, L. R., and D. R. Fulkerson. 1958. Constructing maximal dynamic flows from static flows. *Operations Research*, 6:419–433.

Garey, M. R., and D. S. Johnson. 1979. *Computers and Intractability: A Guide to NP-Completeness*. W.H. Freeman.

Johnson, D. S., and C. C. McGeoch. 1992. DIMACS implementation challenge workshop: algorithms for network flows and matching. DIMACS Tech. Report 92-4.

Operations Research, 41(1), 1993. (Special Issue on Vehicle and Aircraft Routing and Scheduling.)

Orlin, J. B. 1983. Maximum throughput-dynamic networks flows. *Math. Programming*, 27:214–231.

Solomon, M. M. 1988. Time window constrained routing and scheduling problems. *Transportation Science*, 22(1):1–13.

Tardos, É. 1985. A strongly polynomial minimum cost circulation algorithm. *Combinatorica*, 5:247–255.

Zawack, D. J., and G. L. Thompson. 1987. A dynamic space-time network flow model for city traffic congestion. *Transportation Science*, 21(3):153–162.

AUTHOR BIOGRAPHIES

ZHIQIANG CHEN is a doctoral student in the Department of Computer Science at the University of New Mexico, working under the direction of Professor Moret. He earned his BS degree in Computer Engineering from Tsinghua University in September 1991 and his Master's in the same field from the Shenyang Institute of Computing Technology in September 1994.

ANDREW HOLLE earned his Master's in Computer Science from the University of New Mexico under the direction of Professor Moret in May 1995. He earned his BS in Chemical Engineering from the University of Texas (Austin) in 1984 and worked for 8 years in the oil industry, principally in Houston, but also spending two years in Vienna commissioning a tank farm automation system. Upon receiving his MS, he joined the advanced development group of Setpoint in Eindhoven, the Netherlands, working on optimization problems in the refining industry.

BERNARD MORET (`www.cs.unm.edu/~moret`) is Professor of Computer Science at the University of New Mexico. His research interests center on the design and analysis of discrete algorithms and data structures, particularly in practical settings. He has over 30 publications in the area, has graduated 4 PhD students, and directs a research group of 4 students with funding from the Office of Naval Research to investigate aircraft routing problems. He is the editor-in-chief and founding editor of the *ACM Journal of Experimental Algorithmics*, an on-line refereed journal devoted to the practical side of discrete algorithms and data structures.

JARED SAIA is a Master's student in the Department of Computer Science at the University of New Mexico, working under the direction of Professor Moret. He earned his BS in Computer Science from Stanford University in 1993, then spent a year in Japan at the ATR Labs working on research in natural language understanding.

ALI BOROUJERDI earned his BS in Computer Science from the University of Lancaster (England), and his MS and Ph.D. in Computer Science from the University of New Mexico under the direction of Professor Moret. Upon receiving his Ph.D. in December 1994, he joined the staff of the Naval Research Laboratory, Advanced Information Technology Branch, where he conducts research in discrete optimization.