Application of Min-Cost Flow to Airline Accessibility Services

Dan Gulotta
Daniel Kane
Andrew Spann
Massachusetts Institute of Technology
Cambridge, MA

Advisor: Martin Z. Bazant

Summary

We formulate the problem as a network flow in which vertices are the locations of escorts and wheelchair passengers. Edges have costs that are functions of time and related to delays in servicing passengers. Escorts flow along the edges as they proceed through the day. The network is dynamically updated as arrivals of wheelchair passengers are announced.

We solve this min-cost flow problem using network flow techniques such as price functions and the repeated use of Dijkstra's algorithm. Our algorithm runs in an efficient polynomial time. We prove a theorem stating that to find a no-delay solution (if one exists), we require advance notice of passenger arrivals only equal to the time to traverse the farthest two points in the airport.

We run our algorithm on three simulated airport terminals of different sizes: linear (small), Logan A (medium), and O'Hare 3 (large). In each, our algorithm performs much better than the greedy "send-closest-escort" algorithm and requires fewer escorts to ensure that all passengers are served.

The average customer wait time under our algorithm with a 1-hour advance notice is virtually the same as in the full-knowledge optimal solution. Passengers giving only 5-min notice can be served with only minimal delays.

We define two levels of service, Adequate and Good. The number of escorts for each level scales linearly with the number of passengers.

One hour of advance notice is more than enough. Epsilon Airlines can make major improvements by using our algorithm instead of "send-closest-escort" ; it should hire a number of escorts somewhere between the numbers for Adequate and Good service.

The UMAP Journal 27 (3) (2006) 367–385.

Introduction

Annually, 2.2 million disabled people over the age of 65 travel [Sweeney 2004]. Major airlines provide wheelchairs and escorts (attendants) for disabled passengers but struggle to manage costs. We consider how few escorts are needed and how best to route them through the day. The algorithm should be flexible enough to deal with incomplete information about the arrival times of passengers or with an unexpected arrival. We recast the problem as a min-cost flow problem, the daily schedule of the escorts being the network flow.

Terminology and Conventions

- **Job.** The process of an escort picking up a wheelchair passenger *(WP)* at an arrival gate and taking them to their connecting departure gate.
- **Job starting time.** The time when a WP arrives at the airport. If an escort begins a job after the starting time, the WP had to wait before being helped.
- **Job ending time.** The time at which a WP needs to be at the departure gate to board in a timely manner. If an escort cannot finish a job after the ending time, the passenger misses their plane entirely!
- **Vertex** and **Edge.** Our airport is set up as a graph that is used for distance calculations. Our algorithm is set up as a min-cost flow problem on a network that uses a completely unrelated graph.
- **Price**, **Cost**, and **Effective Cost.** We use the standard graph-theory definitions for these words, not business definitions. Price and cost are not the same thing: Cost is the penalty for performing jobs (which is negative, since we want people to perform jobs), while price is an artificial construct that we need to run Dijkstra's algorithm.

Assumptions and Assumption Justifications

About Airports and Flights

- **In our airport graphs, each edge has uniform density of gates.** This is not always the case in real life, but the mathematics of our model would not change if this assumption were removed, only the maps used in its implementation.
- **The passengers' flight connections go between random gates.** Since passengers may be connecting a wide variety of places, the airline cannot optimize for all the distances between departure and arrival gates.

- **Arrival times and gates do not change less than 1 h before arrival; departure times and gates do not change less than 3 h before departure.** We permit planes to be delayed or have gates changed but not while an escort is helping a WP or moving to pick up a WP. This assumption is not fully realistic, but it should not affect our end results too much and it makes the model easier to implement. We can still have the unexpected arrival of WPs.

- **All passengers are making connections, not arriving at their final destinations.** This assumption is unrealistic, but our model could easily be changed to handle final destinations as well, by adding a gate to represent the baggage claim and having passengers depart from this gate.

- **Effective layover times range from 45 min to 132 min**, linearly biased towards shorter layover times. Effective layover time means that we take into consideration that WPs are usually the last to deboard and the first to board. These times are arbitrary, not based on empirical data.

- **WP arrivals are random and uncorrelated.** There is no tendency for such passengers to travel in groups. Although this assumption might be false around the time of the Special Olympics or other specific events, for a typical day it is reasonable.

About Escorts

- **Escorts receive orders via computer.** Communication of orders to escorts is fast and robust, and a computer runs the algorithm.

- **Escorts all walk at the same pace.** A typical modeling assumption.

- **Escorts travel half as fast when a person occupies the wheelchair as with an empty wheelchair.** The factor of two is arbitrary, in the absence of data.

- **The level of traffic in the airport does not affect travel times within the airport.** We make this assumption so that when we analyze our results we avoid confounding the effects of increasing the number of WPs in the airport with the effects of changing the airport layout.

- **Escorts do not abandon WPs.** Regulations prohibit airlines from leaving a passenger in a wheelchair unattended for more than 30 min [U.S. Department of Transportation n.d.]. Such a passenger may still require transportation to restrooms. The escort remains on hand until the passenger boards, when the flight crew takes over. So each escort pushes a single wheelchair that never leaves their sight (hence less concern about loss of equipment).

Cost Structure

We tabulate statistics on the delays in picking up and dropping off WPs for a given traffic load and number of escorts. We graph the delay statistics vs. number of escorts and find the "sweetspot" at which the marginal utility of adding another escort declines.

The term *cost* in this paper is a network-flow definition associated with being late in picking up a passenger, not a direct monetary expenditure. We assign a cost of 1 unit for each minute that a passenger must wait to be picked up upon arriving at the airport. If the passenger cannot be taken to their departure gate at least 15 min before flight time, they miss preboarding and a 30-unit penalty is charged for delaying takeoff of the plane. If the passenger misses a flight, a penalty of 100,000 units is charged.

Simulated Airport Structure

We model the physical structure of the airport as a graph. An edge represents a corridor and a vertex is where corridors meet. Each edge is assigned an amount of time that it takes to transverse it (in minutes, rounded up). We do not distinguish between the left and right sides of a terminal. Some edges are designated as filled with gates from which planes arrive and depart. Escorts can walk only part way down an edge, taking the appropriate fraction of traversal time to do so.

We randomly generate the arrival and departure locations of WPs at arbitrary points on the edges, corresponding to our assumption that gates are spread uniformly along the concourses.

We simulate the following airports: a single straight line, Boston Logan Terminal A (two concourses connected by an underground walkway), and Chicago O'Hare Terminal 3 (Concourses G, H, K, and L).

We precompute the shortest path lengths between each pair of vertices, using the Floyd-Warshall algorithm [Floyd 1962]. This algorithm repeatedly computes $d_k(i,j)$, the minimum path length from i to j passing through only vertices in the set $\{0,1,\ldots,k\}$, for all pairs of vertices i,j. The key insight is that

$$d_k(i,j) = \min\big(d_{k-1}(i,j), d_{k-1}(i,k) + d_{k-1}(k,j)\big).$$

The runtime of this algorithm on a graph with n vertices and m edges is $O(n^3)$.

We use this algorithm for ease of implementation. Later we perform shortest-distance computations on a different network using Dijkstra's algorithm.

Network Flow Definitions and Principles

We give a brief summary of flows [Blum 2005; Demaine and Karger 2003].

Definition. A *network* is a directed graph where each edge is given a positive real valued *capacity* and where two vertices are labeled *source* and *sink*.

Definition. A *flow* is an assignment of nonnegative real numbers to edges such that the number for an edge is at most its capacity; for all vertices other than the source or sink, the sum of values on edges from the vertex equals the sum of values on edges to the vertex. The *size* of a flow is the sum of values on edges from the source minus the sum of on edges to the source.

Definition. A *max-flow* is a flow of maximum possible size for a given network.

We can also assign real-valued *costs* per unit flow to the edges, so that we can talk about the cost of a flow.

Definition. The *cost* of a flow is the sum over all edges of the cost of the edge times the value that the flow assigns to that edge.

Definition. A *min-cost-max-flow* is a max-flow with the smallest possible cost of any max-flow.

If all the edge capacities are integers, there is always a solution that assigns an integer flow to each edge. We will create a data structure to dynamically find such solutions to min-cost-max-flow to decide which escorts should handle which job.

Definition. For a network and a flow on it, the *residual network* is another network on the same vertex set whose edges have capacities equal to the amount of extra flow that could be sent through that edge. Since the flow across some edges can now be decreased, the residual graph has some edges that are the reverse of edges in the original graph. More precisely, the capacity of an edge (i, j) in the residual network is the capacity of (i, j) in the original network plus the flow from j to i minus the flow from i to j. In the residual graph, the cost of (i, j) should be the negative of the cost of (j, i).

The residual graph is useful because *solving min-cost-max-flow on the original graph is equivalent to solving it on the residual graph.*

Definition. A *circulation* is a flow in which the total flow into all vertices (including source and sink) equals total flow out of them.

Definition. A *min-cost circulation* in a network is a circulation of minimum cost.

If f is already a max-flow, the residual graph can accept no more flow. *Hence the problem we must solve is to find a min-cost circulation.*

Definition. A *price function* is a function that associates a price to each vertex.

Definition. The *effective cost* of an edge (i, j) is $c_{i,j} + p_i - p_j$, where $c_{i,j}$ is the cost of (i, j), and p_i and p_j are the prices associated with i and j respectively.

Lemma 1. *The cost of a circulation does not change if we replace costs by effective costs.*

Let $f_{i,j}$ be the flow through edge (i,j) for circulation f. Let $c_{i,j}$ be the cost of (i,j). Let p_i be the price at i. Then the cost of f using effective costs is

$$\sum_{i,j} f_{i,j}(c_{i,j} + p_i - p_j) = \sum_{i,j} f_{i,j}c_{i,j} + \sum_i p_i \left(\sum_j f_{i,j} - \sum_j f_{j,i} \right)$$
$$= \sum_{i,j} f_{i,j}c_{i,j} + \sum_i p_i \cdot 0 = \sum_{i,j} f_{i,j}c_{i,j},$$

which is the cost of the circulation f. □

Hence solving a min-cost circulation problem is equivalent to solving the same problem using effective costs. If all costs are nonnegative, the empty circulation is the min-cost circulation.

Suppose that all of the graph's vertices are reachable from some particular vertex v and all edges have nonnegative cost. We compute shortest paths from v to all other vertices using costs as edge weights, for the following reason: Let w and w' be arbitrary other vertices. If we then *introduce prices* $p_w = d(v,w)$, meaning that the price at w is the distance between v and w, then:

- Since $d(v,w) \leq d(v,w') + c_{w',w}$, the effective cost from w' to w is $c_{w,w'} + d(v,w') - d(v,w)$, which is nonnegative.
- Since there is a path from v to w along which the above inequality is strict, there is a path from v to w of effective cost 0 for all w.

Dijkstra's Algorithm

Our algorithm uses Dijkstra's algorithm every time a new WP arrives or is scheduled to arrive. Dijkstra's algorithm [1959] computes single-source shortest paths in a graph with nonnegative edge weights (i.e., for some vertex v, it computes $d(v,w)$ for all w). The idea is to determine the distances from v in order of increasing distance (a breadth-first search). We use Dijkstra instead of Floyd-Warshall because Dijkstra is more efficient for single-source shortest paths. We approximate $d(v,w)$ and $\min_u(d(v,u)+l_{u,w})$, where $l_{u,w}$ is the length of the edge from u to w and the minimum is over neighbors u of w with known distance from v. The key insight is that the shortest approximate distance is actually correct.

Algorithm Overview

We associate a cost to picking up a job after its starting time, equal to the number of minutes late the job starts, with 30 min more penalty for not transporting a WP to a gate at least 15 min early for preboarding. We associate a cost of effectively infinite magnitude to completely failing to perform a job.

We want at any given time to be able to schedule escorts to complete the currently known jobs at minimal cost. We add jobs to network as the arrival or scheduled arrival of WPs becomes known.

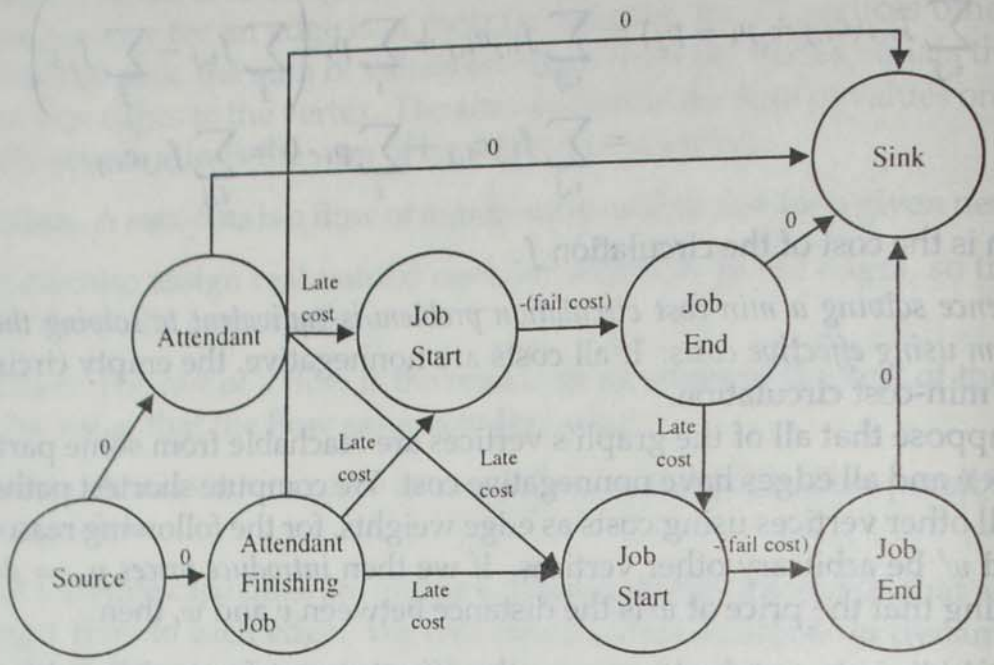


Figure 1. The graph for which we want to solve min-cost flow. Vertices represent jobs, and flow along the edges represents the movement of escorts. Delays are represented as costs, and the edges are labeled with these costs.

We restate the problem as a min-cost-max-flow problem (**Figure 1**). The escorts' schedule is a network flow—of escorts, not WPs. The source is escorts arriving at the start of the day, the sink is them leaving at the end of the day. The vertices are unbusy escorts and the beginning and end of known jobs.

In a min-cost-max-flow problem, we set costs for traversing edges. However, an escort neglecting to pick up a WP involves *not traversing* an edge; we cannot charge for this directly. We equivalently charge a large negative cost of −100,000 for taking the job. At the end of the day, we use this large negative cost to check that all WPs were transported to their flights.

If we knew exactly when each passenger would arrive, we could simply solve min-cost-max-flow on the graph to find the optimal schedule. Instead, we add jobs to our graph as we learn of impending arrivals. We also must delete edges for jobs already performed or that can no longer be performed. We maintain our data structure under the following updates.

Updates in the Algorithm

Time Passing: As time passes, an escort may no longer be able to make it to a job in time; we *update the lateness costs* (edge costs) for this escort. An edge in the current flow corresponds to a job that an escort is trying to get to on time, so its cost does not decrease. For a job that an escort is walking to, the passage of

1 min of time makes the edge cost 1 min more expensive but moves the escort 1 min closer, so these effects cancel. Hence, this update does not decrease any effective cost in our residual graph and maintains our invariants (**Figure 2**).

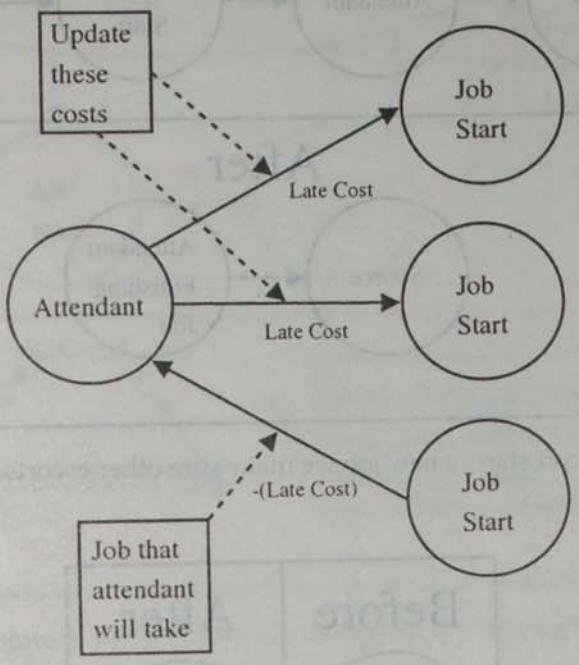


Figure 2. When time passes, we update edge costs.

Taking up a job: When an escort starts work on a job, we *delete the vertices corresponding to the escort and to the start of the job* (along with adjacent edges) and connect the source to the vertex corresponding to the end of the job. Doing this makes for a corresponding change in the residual graph but does not produce negative effective prices (**Figure 3**).

Finishing a job: When an escort finishes a job, the vertex corresponding to the end of that job should be *relabeled* to correspond to the current location of the escort (**Figure 4**). If a new job is assigned, the escort walks toward the job; an escort with no job scheduled does not move.

A new job is announced: A new job is announced when the arrival or predicted arrival time of a WP becomes known. We *create the vertices u and v corresponding to the beginning and end* of the job. Next, we create the edges pointing to u from every end of job and every escort who could make it to this job in time, and edges from v to the sink and to the starts of jobs that could be reached after completing the job given by edge uv. Lastly, we create the edge E from u to v. All edges are given the appropriate costs based on the current time. Assign a high enough price to v and a low enough price to u so that the effective prices of all edges into u or out of v are positive. Then in the residual graph minus E, all effective prices are nonnegative. Our entire graph is accessible from v, because from v we can reach the sink and trace back each of the escorts' schedules to the source. Each job that someone is scheduled to complete can be reached

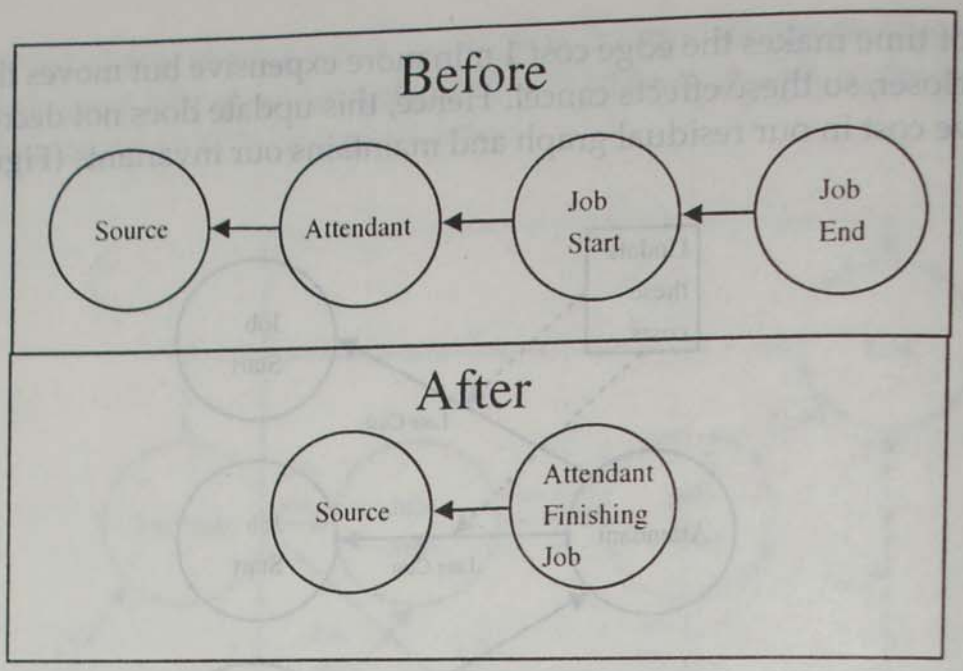


Figure 3. When an escort starts a new job we make sure other escorts don't take the same job.

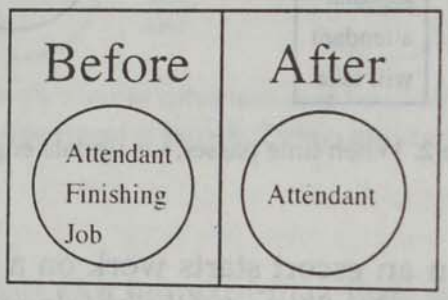


Figure 4. When a job ends, we relabel the job end vertex as an escort vertex.

by tracing back that escort's schedule from the sink. All job start vertices that could be accomplished, but are not scheduled to be, can be reached by getting to the vertex corresponding to an escort who can reach the job and following the edge from to the job's beginning.

Applying the technique above, and adding the size of the smallest cost path from v to w to the price at w, we get nonnegative values for all of these effective costs and a path P from v to u in the residual graph consisting of edges with zero effective cost. If the effective cost of E, is nonnegative, we are done; otherwise, we augment our flow by the cycle PE. This negates the effective cost of all these edges in the residual graph and changes their direction, and hence makes all the effective costs nonnegative (**Figure 5**).

Algorithm Performance

This data structure is fairly efficient. Suppose that there are A escorts and J jobs. The runtime of time passing is $O(AJ)$; that of taking up or finishing a

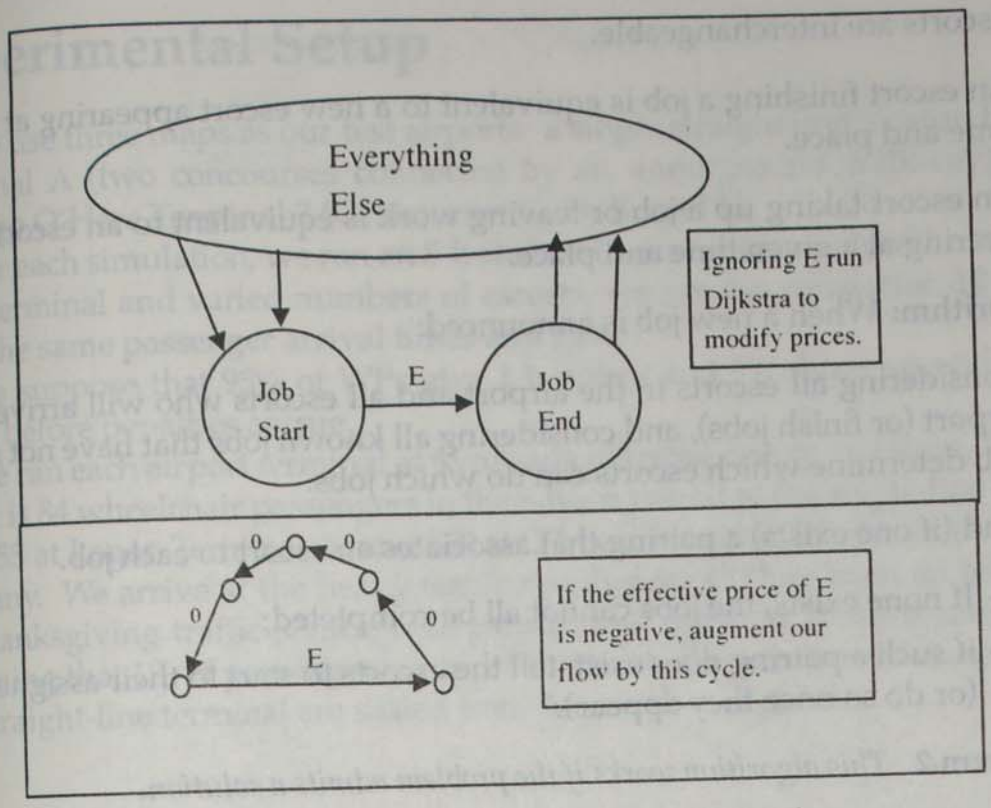


Figure 5. We add new jobs as they are announced.

job is $O(1)$; and learning about a new job depends on the runtime of Dijkstra's algorithm and is bounded by $O((A+J)^2 \log(A+J))$ [Dijkstra 1959].

Our Algorithm Finds Optimal Solutions

We present a theorem that states what classes of algorithms find a solution that transports all passengers to their flights without delays, provided such a solution exists.

We formulate a more general version of the problem as follows. We have some number of escorts; they arrive in the airport at predetermined times of day in predetermined places, but particular escorts are not guaranteed to leave at a particular time. We are required to complete jobs, each requiring that an escort be in a particular location at a given time and remain occupied until released at another known location and time; that end location can be reached from the start location in the time allotted.

All jobs are announced enough ahead of time that an escort could reach the starting location from anywhere in the airport from the time when it was announced. This time is bounded by the time to traverse between the farthest two points in an airport (e.g., 18 min in Chicago O'Hare Terminal 3 with an empty wheelchair; for any airport, advance notice of 30–60 min for arrival of a WP is more than enough).

We stipulate:

- Escorts are interchangeable.
- An escort finishing a job is equivalent to a new escort appearing at a given time and place.
- An escort taking up a job or leaving work is equivalent to an escort disappearing at a given time and place.

Algorithm: When a new job is announced:

- Considering all escorts in the airport and all escorts who will arrive in the airport (or finish jobs), and considering all known jobs that have not started yet, determine which escorts can do which jobs.
- Find (if one exists) a pairing that associates an escort to each job.
  - If none exists, the jobs cannot all be completed;
  - if such a pairing does exist, tell the escorts to start to their assigned jobs (or do so once they appear).

Theorem 2. *This algorithm works if the problem admits a solution.*

[EDITOR'S NOTE: We omit the authors' proof, which uses the marriage lemma [Halmos and Vaughan 1950.]

Our algorithm has an additional property:

If it is possible not to be late for any job nor miss any job, our algorithm will produce such a solution.

Comparison against a Greedy Algorithm

We implemented also a "send-closest-escort" algorithm, which greedily assigns escorts to the closest job according to four rules:

1. When a job becomes known, assign the closest available escort.
2. When a new escort becomes available, assign that escort to the closest available job.
3. Never unassign an escort from a job.
4. Escorts who have nothing to do stay put where they are.

For rules 1 and 2, jobs are not assigned until a fixed time before the arrival time of the passenger (the time to traverse the farthest two points).

Experimental Setup

We use three maps as our test airports: a single straight line, Boston Logan Terminal A (two concourses connected by an underground walkway), and Chicago O'Hare Terminal 3 (Concourses G, H, K, and L).

For each simulation, we ran an 8-h shift with time increments of 1 min. For each terminal and varied numbers of escorts, we ran the simulation 10 times with the same passenger arrival times and gates.

We suppose that 95% of WPs give 1 h notice and 5% show up with only 5 min before penalties accrue.

We ran each airport terminal at two loads of traffic, heavy and light. Heavy traffic is 84 wheelchair passengers in the 8-hour period at the single-line terminal, 155 at Logan Terminal A, and 550 at O'Hare Terminal 3; light traffic is half as many. We arrive at the heavy traffic number for O'Hare from an estimate of Thanksgiving traffic [Chicago Department of Aviation 2001], scaling, and assuming that 1% of passengers need wheelchairs; the numbers for Logan and the straight-line terminal are scaled from the number of gates.

Results

In **Figures 6–7**, the numbers of escorts are plotted as data points; each data point is the average of 10 simulations. The average wait times are the total wait time divided by the number of passenger served. Missed passengers do not affect the average wait time, since their wait time is effectively all day. Results in the lower left-hand corner are more desirable. Any deviation from zero on the x-axis is bad because some passenger was outright ignored!

The results corresponding to the perfect-knowledge solution are almost exactly covered by the corresponding results for our algorithm.

Although the "send-closest-escort" algorithm is frequently competitive for average wait times, it is significantly worse in terms of jobs missed. It scales roughly linearly in balancing delay time and number of jobs missed; it does not perform well without a large number of escorts.

We define two service levels, with corresponding numbers of escorts:

- *Adequate:* The average number of missed passengers in each scenario is lower than 1.
- *Good:* Everyone is served, with an average wait time under 15 min.

In **Table 1**, we summarize the minimum number of escorts needed to reach each service level.

For each airport, the difference between the Good and Adequate service levels is roughly a factor of two, with slightly increasing returns to scale; with larger scales, the staff are spread more uniformly, so it is less likely that a job will crop up with nobody close enough to take it.

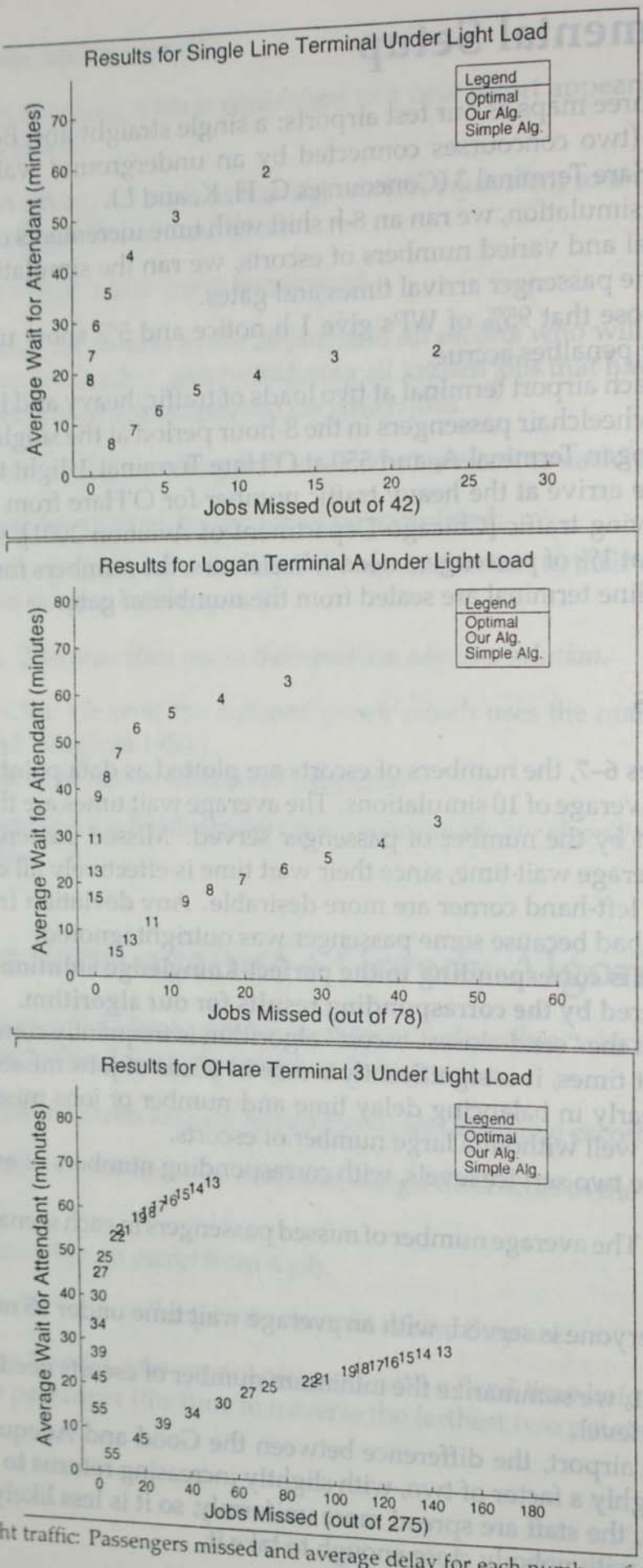


Figure 6. Light traffic: Passengers missed and average delay for each number of escorts, for each airport.

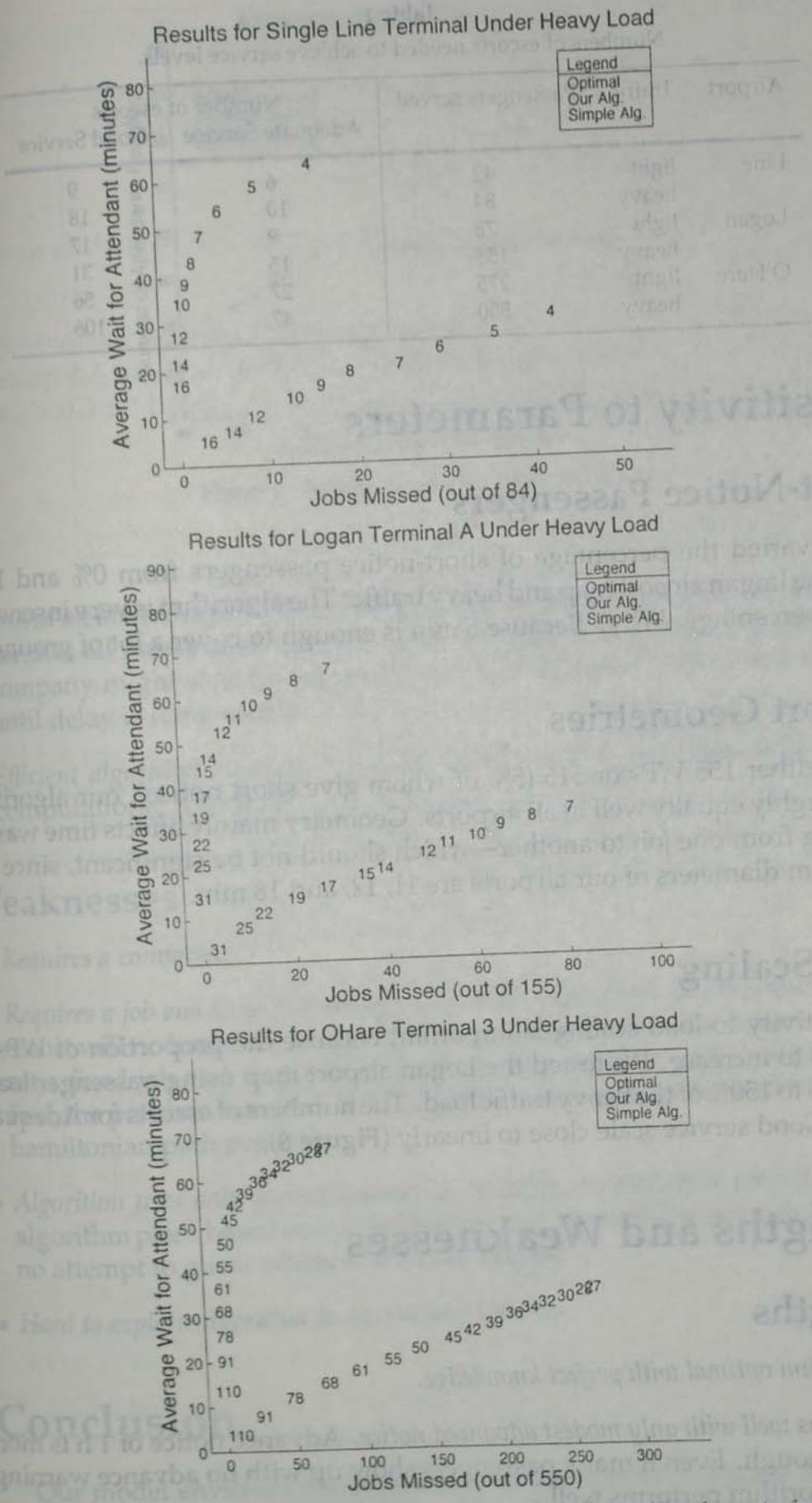


Figure 7. Heavy traffic: Passengers missed and delays for each number of escorts, for each airport.

Table 1.
Numbers of escorts needed to achieve service levels.

Airport	Traffic	Passengers served	Number of escorts	
			Adequate Service	Good Service
Line	light	42	6	9
	heavy	84	10	18
Logan	light	78	9	17
	heavy	155	15	31
O'Hare	light	275	27	56
	heavy	550	47	106

Sensitivity to Parameters

Short-Notice Passengers

We varied the percentage of short-notice passengers from 0% and 100% using the Logan airport map and heavy traffic. The algorithm is very insensitive to this percentage. Why? Because 5 min is enough to cover a lot of ground.

Airport Geometries

For either 155 WPs or 515 (5% of whom give short notice), our algorithm does roughly equally well in all airports. Geometry mainly affects time wasted traveling from one job to another—which should not be significant, since the maximum diameters of our airports are 11, 12, and 18 min.

Load Scaling

Sensitivity to load scaling is important because the proportion of WPs is expected to increase. We tested the Logan airport map using passenger loads from 50% to 150% of the heavy traffic load. The numbers of escorts for Adequate and for Good service scale close to linearly (**Figure 8**).

Strengths and Weaknesses

Strengths

- *Algorithm optimal with perfect knowledge.*
- *Performs well with only modest advanced notice.* Advance notice of 1 h is more than enough. Even if many passengers show up with no advance warning, our algorithm performs well.

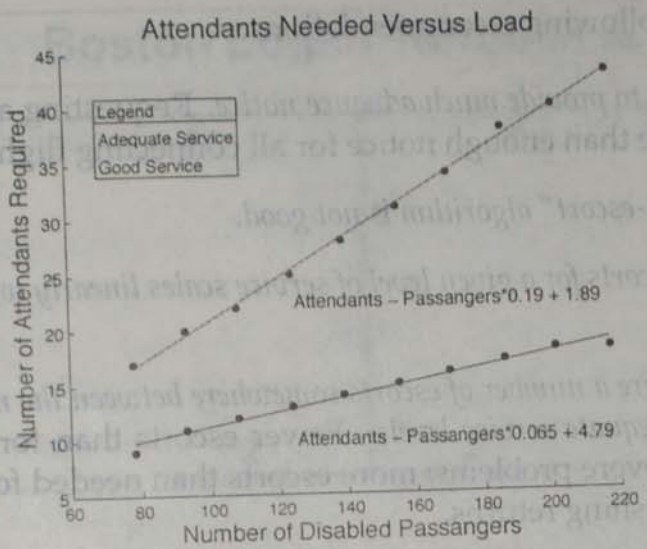


Figure 8. Scaling of escorts as WPs increase.

- *Proved an interesting theorem.* Given sufficient advance notice, our algorithm handles all passengers with no delay if doing so is possible. In practice, a company might stop hiring escorts when delays are small rather than hire until delay is nonexistent.
- *Efficient algorithm runtime.* Our algorithm runs in real time with modest computational resources.

Weaknesses

- *Requires a computer.*
- *Requires a job end time to be specified.* For the algorithm to compute when an escort will no longer be occupied and assign orders into the future, we must set a fixed job end time before starting the job. If an end time is unspecified, the problem becomes NP-complete, because it can be used to solve the hamiltonian path problem.
- *Algorithm uses only factual knowledge of today, no statistical projections.* Our algorithm plans based on current knowledge of scheduled arrivals but makes no attempt to guess where a WP may appear.
- *Hard to explain algorithm to nonmathematicians.*

Conclusion

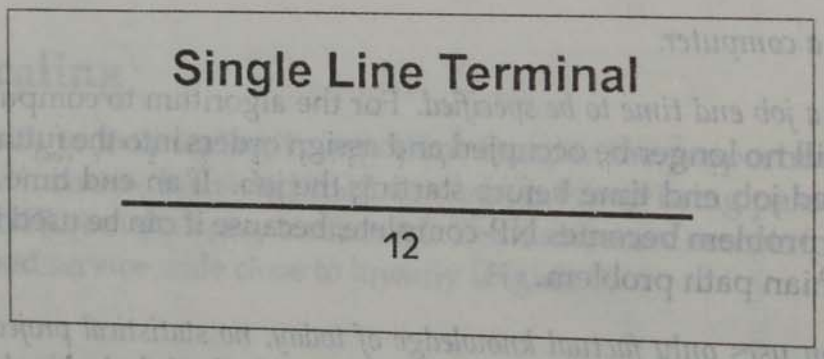
Our model envisions the problem in terms of flow of escorts through the work day. We present an algorithm with compelling theoretical basis for producing good results.

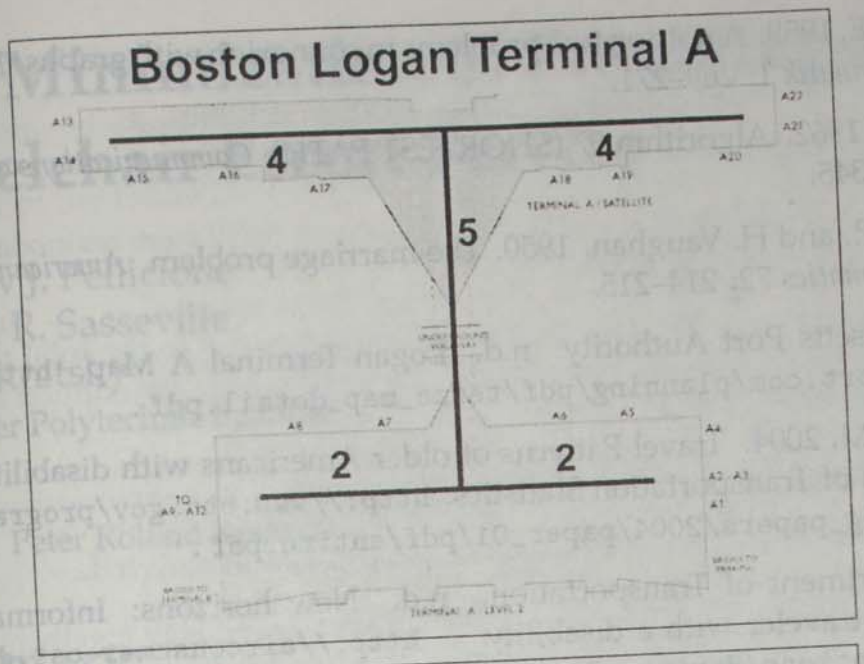
We make the following recommendations:

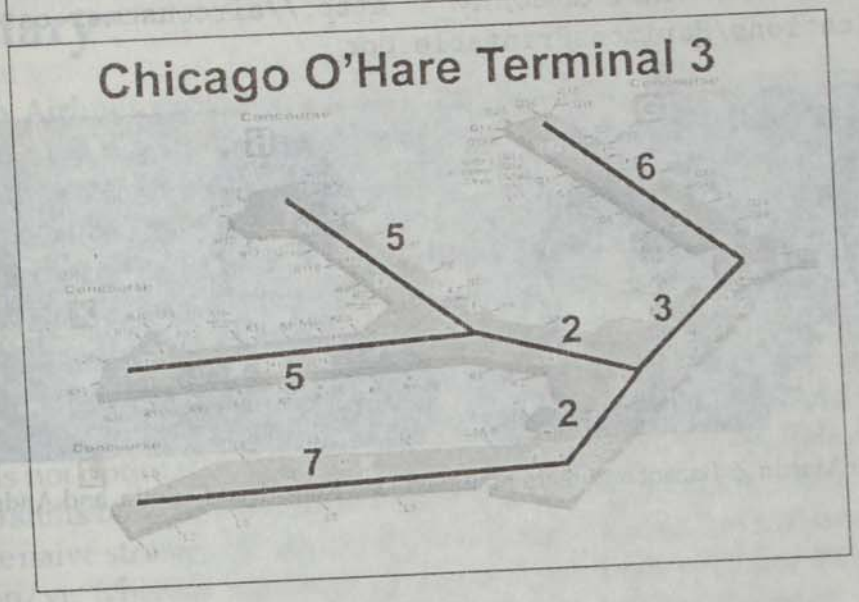
- *WPs do not need to provide much advance notice.* Requesting assistance at first check-in is more than enough notice for all connecting flights.
- *The "send-closest-escort" algorithm is not good.*
- *The number of escorts for a given level of service scales linearly with the number of passengers.*
- *Airlines should hire a number of escorts somewhere between the numbers required for Good and Adequate service levels.* Fewer escorts than for Adequate service results in severe problems; more escorts than needed for Good service produces diminishing returns.

Appendix A: Terminal Maps

These are the graphs for our small, medium, and large simulated airports. A large boldface numbers on an edge is the length of time to traverse the edge; these numbers should be doubled for an occupied wheelchair. The Logan and O'Hare graphs are superimposed on digitally edited versions of the official terminal maps [Massachusetts Port Authority n.d.; City of Chicago n.d.]. The gate positions in the single-line terminal are those used in our simulation; those on the other two maps are the gate positions in real life. In our simulation, gates occur every minute of walking distance along the edges except for edges that are transportation between concourses.







References

Blum, A. 2005. 15-451 Algorithms Course Notes. 10/25/2005. http://www.cs.cmu.edu/afs/cs/academic/class/15451-f05/www/lectures/lect1025.txt.

Chicago Department of Aviation. 2001. The Chicago Airport System prepares passengers for Thanksgiving holiday travel. (16 November 2001). http://www.flychicago.com/doa/avi_news/doa_avi_news_pr_73.shtm.

City of Chicago. n.d. O'Hare Terminal 3 Map. http://www.chicagoairports.com/ohare/terminals/D3.pdf.

Demaine, E., and D. Karger. 2003. 6.854 Advanced Algorithms Course Notes. 9/24/2003. http://theory.lcs.mit.edu/classes/6.854/05/scribe/max-flow-dff.ps.

Dijkstra, E. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik* 1: 269–271.

Floyd, R. 1962. Algorithm 97 (SHORTEST PATH). *Communications of the ACM* 5 (6): 345.

Halmos, P., and H. Vaughan. 1950. The marriage problem. *American Journal of Mathematics* 72: 214–215.

Massachusetts Port Authority. n.d. Logan Terminal A Map. `http://www.massport.com/planning/pdf/terma_map_detail.pdf`.

Sweeney, M. 2004. Travel Patterns of older Americans with disabilities. U.S. Bureau of Transportation Statistics. `http://www.bts.gov/programs/bts_working_papers/2004/paper_01/pdf/entire.pdf`.

U.S. Department of Transportation. n.d. New horizons: Information for the air traveler with a disability. `http://airconsumer.ost.dot.gov/publications/HorizonsPrintable.doc`.

Team advisor Martin Z. Bazant and team members Dan Kane, Dan Gulotta, and Andrew Spann.