A Simulation-Driven Approach for a Cost-Efficient Airport Wheelchair Assistance Service

Samuel F. Feng
Tobin G. Isaac
Nan Xiao
Rice University
Houston, TX

Advisor: Mark P. Embree

Summary

Although roughly 0.6% of the U.S. population is wheelchair-bound, the strain of travel is such that more than twice that amount relies on wheelchairs in airports [Haseltine Systems Corp. 2006].

Two issues have the greatest impact on the cost and effectiveness of this service: the number of wheelchairs and how they should be deployed. The proper number of escorts and wheelchairs is not only a question of the airport but of the volume of passengers, which can vary greatly. If escorts determine their own movements within the airport, lack of coordination could result in areas being unattended; however, fluctuation in requests could be so great that a territory-based plan could overwork some escorts and underwork others.

We present an algorithm for scheduling of the movement of escorts that is both simple in implementation and effective in maximizing the use of available time in each escort's schedule. Then, given the implementation of this algorithm, we simulate the scheduling of requests in a given airport to find the number of wheelchair/escort pairs that minimizes cost.

Methods And Assumptions

We propose a stochastic simulation-driven optimization procedure. We partition the problem into three categories: pre-simulation processing, simulation rules and dynamics, and optimization. The pre-simulation phase generates the necessary inputs for the simulation phase, such as the airport layout and a master passenger request list containing the wheelchair assistance requests for a single day. The simulation phase consists of a continuous event-based model of passenger arrival/departure and wheelchair/escort movement. Finally, we minimize costs over the number of escorts.

Pre-Simulation

Airport Layout

The simulation is customized for the layout of each airport. We represent an airport as a bidirectional graph, in which nodes indicate gates, entrance/exit points, or other places of similar interest. Edges between nodes indicate travel paths, usually through the main hallway of a concourse. We constructed the graphs with the aid of satellite images [Google.com n.d.].

Passengers must travel through long corridors to reach departure gates. A typical concourse has gates on either side of the main corridor. We assume that the time required to travel from any gate to any gate is substantial; that is, even if two gates are adjacent or on opposite sides of the main corridor, we assess the travel time between the two gates.

The graphical representation of the airport is encoded in an $n \times n$ adjacency matrix A, with entry $A_{i,j}$ denoting the travel time between location i and location j. We determine travel times by figuring the actual distance divided by a walking speed of 3 mph. The shortest possible travel times (calculated using Dijkstra's algorithm [Wang 2006]) from every location to every other location is referenced in matrix D, with entry D_{ij} denoting the shortest travel time between nodes i and j.

We assume that the escorts know the shortest path between any two gates, because they are familiar with the airport environment. In our simulations we do not consider the distance between the gate and the airplane.

Wheelchairs And Escorts

We treat a wheelchair and its escort as a single traveling entity, the "escort." The airline may have additional wheelchairs on hand in the event of a malfunction, and we incorporate the cost of the additional wheelchairs into the maintenance and storage costs of wheelchairs in operation. The escort's job is to travel to the arrival gate of the passenger and transfer the passenger to the departure gate.

An important assumption is that the number of escorts remains constant throughout the simulation period. In reality, escorts rotate in shifts; but with a simulation period of one day, we assume that escorts starting a shift immediately replace escorts ending one. Similarly, during the simulation we do not allow hiring or firing of workers, nor buying or breaking of wheelchairs; instead, we represent the costs associated with these actions with a sunken cost term in the total cost function.

Passenger Request List

Given a terminal, we create a flight schedule for one day. We look at the total number of passengers who pass through the airport in one day. We estimate the average load of a plane to be 125 passengers, which we use to estimate the number of flights arriving or departing at the terminal in one day. Observation of departing flight information at a busy airport [City of Atlanta 2006] confirms the information in another source [Neufville 1976]: There is regular activity between 6 A,M, and 10 P.M. and relatively few flights at night. We therefore space our departures evenly between these times, and then perturb these values by a random shift of less than an hour, so that we are certain not to test our algorithm against just one schedule. Subsequently, these flights are assigned to specific gates.

Next, we create the requests for the day. We generate the number of requests based on the passenger volume that we are trying to mimic and the percentage of the population that requires wheelchair assistance. For different runs, this value was either 0.6% or 1.2% [Haseltine Systems Corp. 2006]. Each request is assigned an arriving flight and a departing flight, with the assumption that no layover of less than half an hour should be attempted. We assume that a certain percentage of wheelchair passengers have phoned the airline ahead of time; time of request is set to 0. For the remainder, we generate random request times, varying from more than five hours to a half hour before arrival of the wheelchair passenger. We sort this list by request time, so that when the algorithm descends the list, it mimics a dispatcher's receiving the requests at varying times throughout the day, including the wheelchair needs that occur with little notification.

Different daily scenarios can be modeled by altering the generation of the request list. The request list models the passenger traffic load throughout the simulation, since it contains a greater concentration of requests during peak hours of operation. Furthermore, request frequency throughout the day can be increased to reflect operation during holiday-travel seasons at hub airports, or yearly peak-travel at airports in popular vacation destinations.

Scheduling Plan

We assume that escorts communicate with a dispatcher via a walkie-talkie, and that the dispatcher has a schedule for each escort.

An escort who has completed a task calls the dispatcher to find out the next. We assume that the dispatcher knows how long it takes an escort to get between two points x and y, which we call δ_{xy}.

A dispatcher receives requests at varying times throughout the day. Each request contains four pieces of information: the time and location of the passenger's arrival, t_a and a, and the time and location of departure, t_d and d. For the passenger to reach the destination in time, the escort must arrive at a location by some final time,

$$t_f = t_d - \delta_{ad}.$$

The algorithm's version of "first come, first served" is that one task cannot replace another on a schedule if its final time is later. This keeps every schedule compact: If a switch is made, the only result is the new task starting sooner after the previous one (switching will be discussed shortly).

To determine which escort should be assigned a passenger, the dispatcher first finds out if anyone can complete the task. From those who can, he selects one who can complete it in an optimal way, as follows.

The worst way in which a request can be fulfilled is for the escort to bring the passenger to his destination late yet within the window δ_w in which the flight is delayed. To determine if escort e can do this, the dispatcher looks at what location l_e the escort will be at completion of the last job before the final time of the request, and when that job will be completed, t_e. We need

$$t_e + \delta_{ea} < t_f + \delta_w.$$

An escort who meets this requirement is in group O_1.

A better way in which a request can be fulfilled is if an escort can bring the passenger to the destination on time but by removing another passenger from his schedule later. The condition for this one is the same as above, but without the delay term:

$$t_e + \delta_{ea} < t_f.$$

An escort who meets this requirement is in group O_2.

Because we want to do as little reshuffling of schedules as possible, an even better situation is for an escort to be able to take on a request but have to push back the time of completion of later tasks without forcing any late departures. An escort who meets this requirement is in group O_3.

Finally, the ideal situation is when the dispatcher can assign a request to an escort without rescheduling that escort's later tasks. If the next task is to start at time t_s at location s, then escort e must be able to complete the request with enough time to go from d to s by t_s,

$$t_e + \delta_{ds} < t_s.$$

An escort who meets this requirement is in group O_4.

The dispatcher determines the most optimal group that is not empty and chooses an escort in it for the request whose previous task brings that escort closest to the arrival location of the passenger in the new request. If a new request bumps out one or more queued requests, they must be rescheduled before the dispatcher advances to to the next request on the list.

If a request cannot be scheduled, every escort will either be busy with another request or will be too far away to arrive in time. In such a case, the passenger must be scheduled for a later flight and/or compensated for missing the flight, and the situation falls out of the algorithm.

Simulation Algorithm

We envision the problem as a temporal "packing" problem—the dispatcher must fit as many tasks onto the escorts' schedules as possible.

```
Algorithm : MainSimulation($D, R, N_E$)

create escort task array $E$
$I = $ FindIndexUnfulfilledRequest($R$)
while $I > 0$
    do { $O = $ MakeOptimalityMat($D, R_I, E$)
       { comment: Now execute request
       { $(R, E) = $ ExecuteRequest($D, R, E, I, O$)
       { $I = $ FindIndexUnfulfilledRequest($R$)
$totalmissed = $ Sum($R_{\text{missed flights}}$)
$totaldelay = $ Sum($delaytime$)
return $(totalmissed, totaldelay)$
```

MainSimulation() handles the entire simulation. We input the shortest-travel-time matrix D, list of requests R, and number of escorts N_E. Entry E_j of the task array is escort j's task schedule. The two main routines within the simulation are MakeOptimalityMat() and ExecuteRequest(). Together, they determine which escort is most suited to be assigned the request at hand, and how the current schedule of the escort will be changed to accommodate the new request. MakeOptimalityMat() makes a $N_E \times 4$ matrix, with row j representing the inclusion in or exclusion from the optimality groups of escort e_j.

Each row of MakeOptimalityMat() is generated by OptimalityCheck(), shown on the next page.

For each request, OptimalityCheck() assigns escorts into optimality groups. OptimalityCheck() creates a row of an optimality matrix O, with entry $O_{i,j}$ denoting whether escort i is *group j-optimal*—that is, i is in group j.

Finally, ExecuteRequest() (shown on p. 401) assigns an escort, if possible.

As the groups descend in optimality, the dispatcher undertakes more and more actions in attempting to assign the request at hand.

Algorithm : OPTIMALITYCHECK(D, R_I, e_j)

```
initialize $O_j = (0,0,0,0)$
if Escort $j$ is Group 1 Optimal (can fulfill $R_I$ with delay)
  then $O_{j,1} = 1$
if Escort $j$ is Group 2 Optimal(can fulfill $R_I$ w/o delay)
  then $O_{j,2} = 1$
if Escort $j$ is Group 3 Optimal(can fulfill $R_I$ w/o removing tasks)
  then $O_{j,3} = 1$
if Escort $j$ is Group 4 Optimal(can fulfill $R_I$ w/o shifting tasks)
  then $O_{j,4} = 1$
return $(O_j)$
```

Deployment Plan

We measure costs in United States Dollars (USD). There are two cost categories:

- costs dependent on the number of wheelchair/escort pairs, specifically, wheelchair maintenance/storage costs and escorts' wages, and
- costs dependent on the number of delayed flights and passengers who miss their flights.

All wheelchair and escort costs are considered fixed throughout the (one-day) duration of a simulation.

In our cost function, N_E is the number of escorts, R is the daily request list, D is the airport layout, X is the number of missed flights, and Y is total amount of time that flights that must be held at the gate due to late passengers. The objective function is thus:

$$[X, Y] = \text{MAINSIMULATION}(D, R, N_E),$$
$$C(X, Y) = \text{Cost}_{\text{fixed}} + \text{Cost}_{\text{miss}} X + \text{Cost}_{\text{delayed}} Y$$

The average cost per hour of a flight held at the gate is assumed to be \$1,018, the average cost in 1986 [Nathan L. Kleinman, Stacy D. Hill 1998] adjusted for inflation [Friedman 2006]. The cost of missing a flight is \$500, the price of ticket reimbursement and/or lodging if necessary. Our fixed costs include the maintenance costs of the wheelchairs and wages of escorts. We assume that wheelchairs cost \$130/yr/chair [Alexander 2006] to maintain, store, and replace as needed, and that escort wages are \$10/h. [Avjobs.com 2006]. The

Algorithm : ExecuteRequest(D, R, E, I, M)

```
if column 4 of $O$ contains a 1
  then { Find escort with closest previous task $e^*$
         Insert task into schedule of $e^*$
         Mark request $R_I$ as fulfilled
  else if column 3 of $O$ contains a 1
  then { Find escort with closest previous task $e^*$
         Determine insertion spot $s$ for task
         Move jobs after $s$ back farthest possible
         Insert task into schedule of $e^*$
         Mark request $R_I$ as fulfilled
         Move jobs after $s$ forward farthest possible
  else if column 2 of $O$ contains a 1
  then { Find escort with closest previous task $e^*$
         Determine insertion spot $s$ for task
         Move jobs after $s$ back farthest possible
         Insert task into schedule of $e^*$
         Mark request $R_I$ as fulfilled
         Remove overlapping tasks in schedule
         Mark corresponding requests as unfulfilled
         Move remaining jobs after $s$ forward farthest possible
  else if column 1 of $O$ contains a 1
  then { Find escort with closest previous task $e^*$
         Determine insertion spot $s$ with minimum delay
         Move jobs after $s$ back farthest possible
         Insert task into schedule of $e^*$
         Mark request $R_I$ as fulfilled
         Remove overlapping tasks in schedule
         Mark corresponding requests as unfulfilled
         Move remaining jobs after $s$ forward farthest possible
         Log delay of $R_I$
  else { Mark task as fulfilled
         Log $R_I$ as missed flight
return $(E, R)$
```

$\text{Cost}_{\text{fixed}}$ term is thus given by

$$\text{Cost}_{\text{fixed}} = \left(10 + \frac{130}{365 \times 24}\right) H N_E$$

where H is the simulation period in hours.

Results and Analysis

Test Protocol

Before delving into the results of test simulations, we clarify certain protocols used.

- Each data point on every plot represents 10 trials.
- Error bars are one standard deviation.
- Half of all requests are known ahead of passenger arrival (typical).

Optimizing Schedules

We demonstrate the effectiveness of adaptive scheduling, that is, the allowing of escorts to be placed in optimality groups 1, 2, and 3. We simulated a small one-concourse setting with moderate traffic (15,000 passengers/day) and successively removed placing escorts in optimality groups 1, 2, and 3, in that order. In other words, the simplest algorithm is for a request to be missed if no escort is in O_4, the next simplest algorithm is for a request to be missed if no escort is in O_4 or O_3, etc. These simulations were carried out assuming that 1.2% of the passengers require wheelchair assistance.

There are large gains from shifting around existing tasks in an escort's schedule, that is, the inclusion of group O_3. The effect is substantial—we avoid hiring five or six escorts, resulting in savings of about \$1,000/d (**Figure 1**). Savings would increase with passenger traffic and airport size. With several airports, Epsilon Airlines may see total savings on the order of \$10,000/d, merely by adopting our adaptive scheduling.

Performance/Sensitivity across Concourses

We use Chicago O'Hare terminals as test locations for examining performance in different-sized settings. We ran simulations for one-, two- and four-concourse settings, corresponding to **Figures 2—4**. Each set of simulations spans three levels of terminal traffic, assuming that 0.6% of passengers require wheelchair assistance. Comparing costs vs. number of escorts, we observe

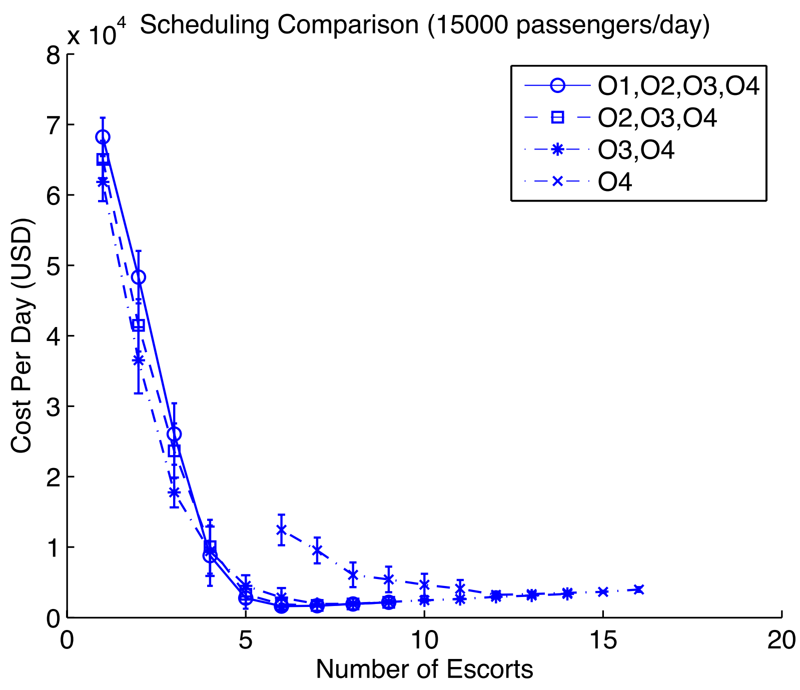


Figure 1. Comparison of scheduling methods.

an increase in the optimal number of escorts roughly proportional to the increase in the number of concourses. Furthermore, as airport traffic increases, we generally see a rise in the number of escorts needed to maintain optimal cost. Figure 4 shows that in a single-concourse setting, increased traffic density barely increases the optimal number of escorts, whereas **Figures 3** and **4** show substantial increases in escort demand within the two- and four-concourse settings, due to longer travel times between gates.

Performance/Sensitivity across Airports

Our algorithm has the flexibility to address any airport configuration, since we use satellite images of an airport to convert it to its node-edge representation.

We perform a comparison analysis on three airports. We examine two-concourse sections from New York's LaGuardia (LGA), Chicago O'Hare (ORD), and Dallas/Fort Worth (DFW) airports, at two separate traffic levels, assuming that 1.2% of passengers require wheelchair assistance. From the satellite images, we hypothesized that DFW would fare the worst, due to passengers having to walk from one side of the circular terminals to the other (almost a mile), and that Chicago O'Hare would perform best, since it has a more compact layout, with passengers able to cross through a triangular intersection among concourses.

However, **Figure 5** shows that algorithm performance is statistically equivalent, despite different airport geometries, over the wide range of numbers of passengers per day, and over 10 averaged trials for each data point. Our al-

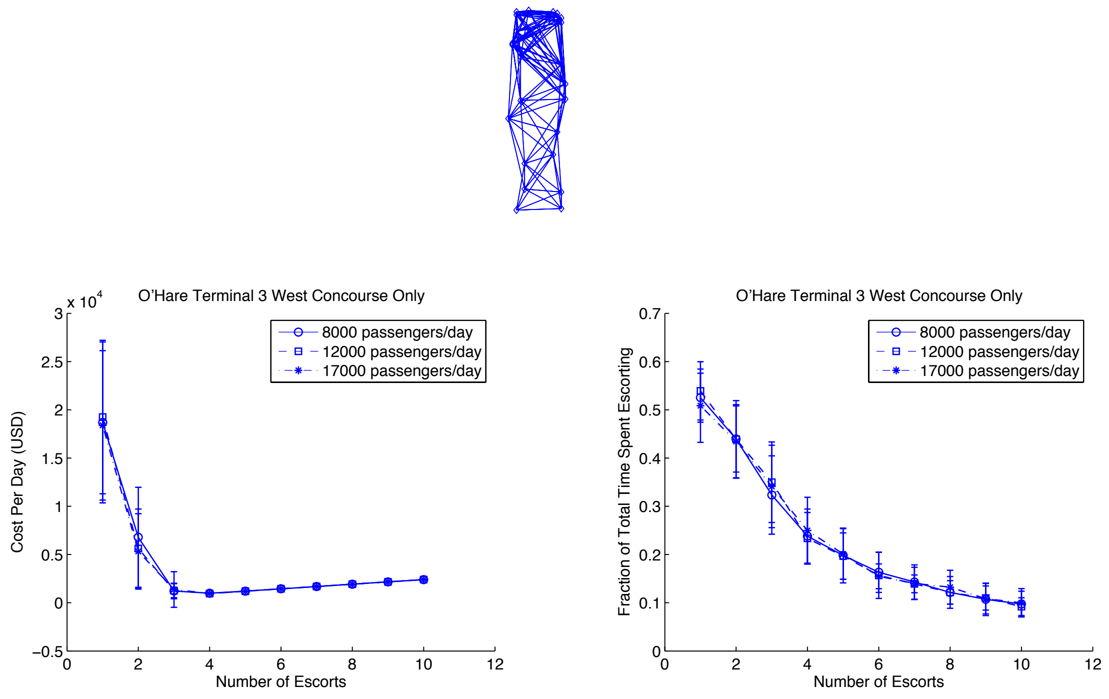


Figure 2. Chicago O'Hare Terminal 3 West Concourse only.

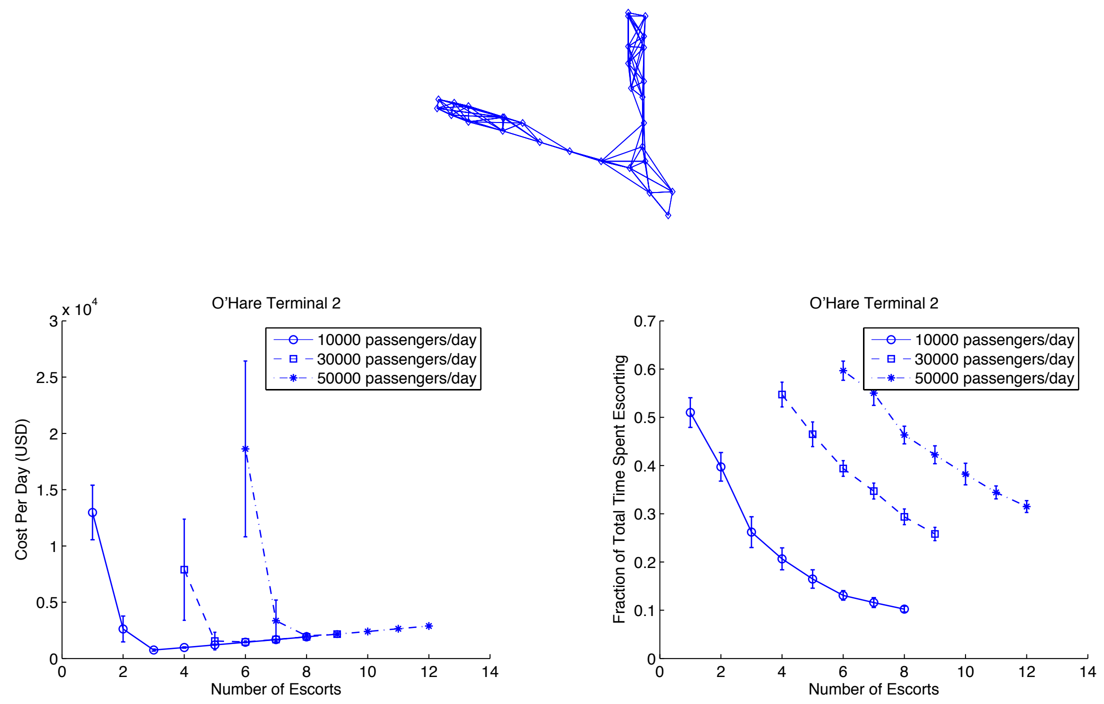
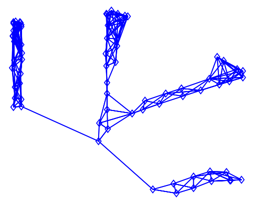


Figure 3. Chicago O'Hare Terminal 2.



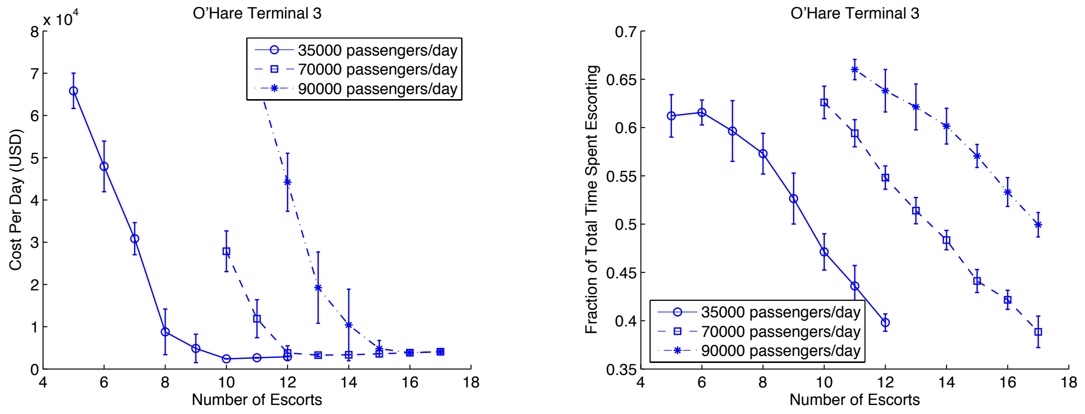


Figure 4. Chicago O'Hare Terminal 3.

gorithm performs equally well regardless of airport configuration, though for higher traffic levels we might see differences among airport geometries.

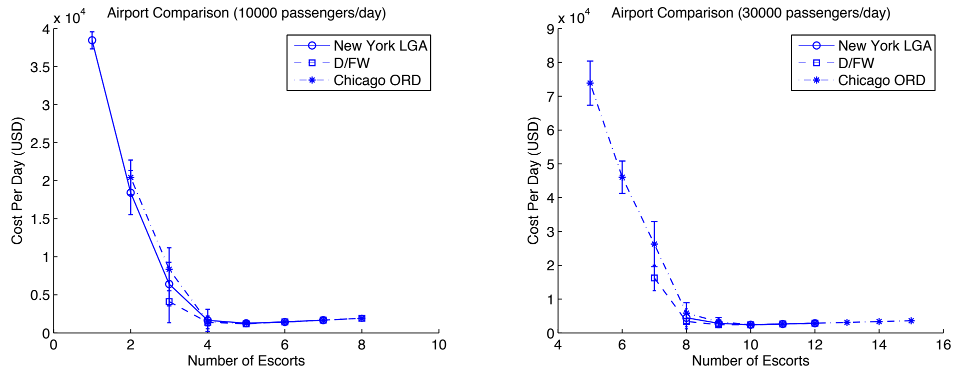


Figure 5. Comparison of algorithm performance at different airports.

Predicting For An Aging Population

The demand for wheelchair assistance will increase in the future. The question is, What does this entail? Should an increase of 10% in the requests per capita be treated the same as a 10% increase in total volume of passengers? The answer from our simulations appears to be that it should be treated as more. A 10% percent increase in population would increase the number of scheduled flights per day; a 10% increase in requests per capita would not. In other words, as the average passenger ages, the result is not just more requests but more requests per plane.

We ran two series of simulations for a two-concourse airport.

- We increased the number of passengers (and thus correspondingly the number of flights) from 33,000 to 42,000.
- We held the number of passengers constant at 30,000 and increased the request percentage from 1.32% to 1.68%.

In each series, the number of requests is the same, but the average minimum cost is greater when the percentage increases (**Figure 6**).

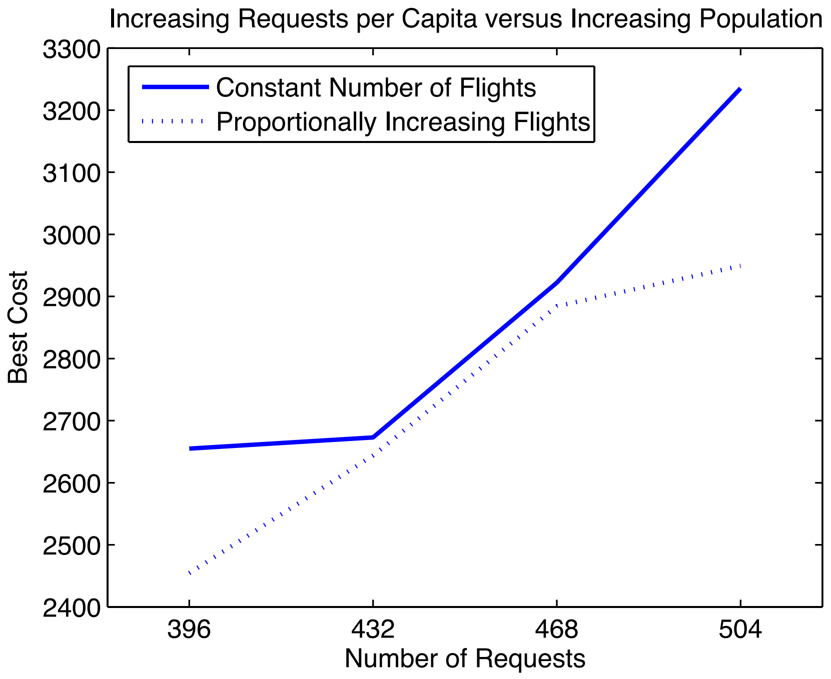


Figure 6. Comparison of increase in requests.

Conclusions

A weakness of our approach is that we do not specifically optimize towards minimizing damages. For example, an escort may have five short tasks (in terms of time from arrival gate to departure gate) queued, when a lengthy request is received. The dispatcher may end up striking three or four tasks from the escort's schedule in order to fulfill the one new task—but the removed tasks' passengers may miss their departure times. We sacrifice many passengers for the sake of one. However, this situation is rare, since we try to minimize the chance of any passenger missing his/her flight. If this type of situation is encountered frequently, then we have probably not optimized the number of escorts.

Our method also assumes no delayed arrivals or departure delays due to other causes.

The dispatcher has the difficult task of managing escorts' schedules, but this is feasible with a computer.

These weaknesses are outweighed by the demonstrated robustness and improvements of our approach. We have shown in a variety of settings that not only does our algorithmic approach work but it outperforms simpler algorithms by substantial margins.

References

Alexander, Richard. 2006. Lifecare planning for the BK amputee: Future medical costs. `http://consumerlawpage.com/article/amputee.shtml`.

Avjobs.com. 2006. Aviation career salary ranges. `http://www.avjobs.com/table/airsalry.asp`.

City of Atlanta. 2006. Atlanta international airport flight information. `http://www.atlanta-airport.com`.

Friedman, S. Morgan. 2006. Inflation calculator. `www.westegg.com/inflation/`.

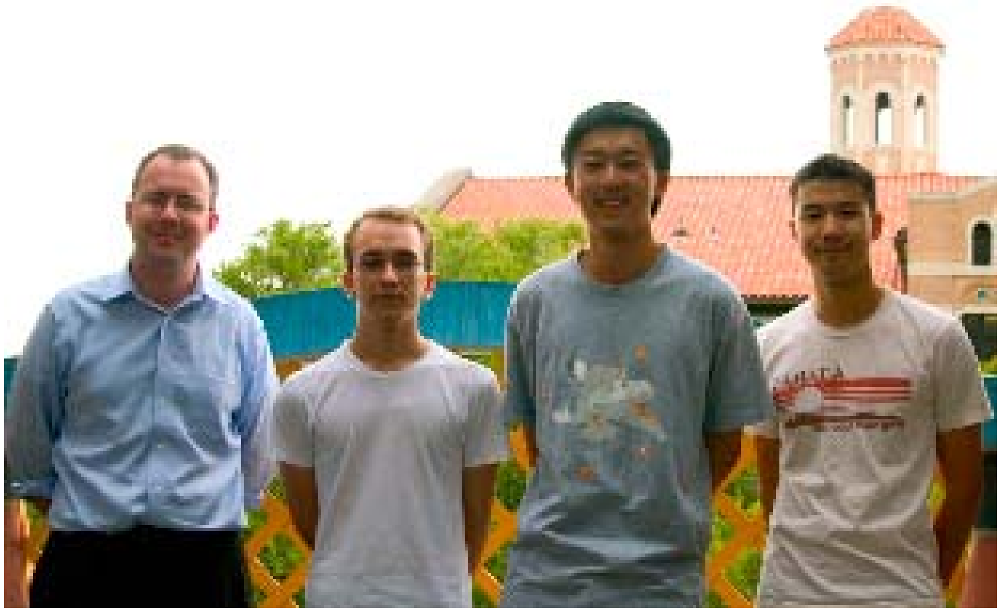
Google.com. n.d. Google maps. `http://maps.google.com`.

Haseltine Systems Corporation. 2006. Haseltine Systems Corporation. `http://www.haseltine.com/data.html`.

Kleinman, Nathan L., Stacy D. Hill, and Victor A. Ilenda. 1998. Simulation optimization of air traffic delay cost. In *Proceedings of the 1998 Winter Simulation Conference*, edited by D.J. Medeiros, E.F. Watson, J.S. Carson, and M.S., Manivannan, 1177–1181.

Neufville, Richard De. 1976. *Airport Systems Planning*. Cambridge, MA: MIT Press.

Wang, Yi. 2006. Dijkstra algorithm consistent with cyclic paths. `http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=7869&objectType=file`.


Team advisor Mark Embree and team members Toby Isaac, Nan Xiao, and Sam Feng.