Advanced Aircraft Sequencing: Stacking the deck to deal you a better hand

Vivek Kumar
David Teale
Jianfeng Wang
Seth Wenchel, Team Lead

Sponsors:
Dr. Lance Sherry
Dr. John Shortle
Center for Air Transportation Systems Research
George Mason University

Table of Contents

I. Executive Summary 3
II. Introduction 4
A. Background 4
B. Problem Statement 5
III. Modeling Approach 5
A. Performance Metrics 5
B. Assumptions 6
C. Models 6
1. First Come, First Serve (FCFS) 8
2. Passenger Delay Minimization, P_delay 8
3. Vehicle Delay Minimization, V_delay 9
4. Vehicle Throughput Maximization, V_ thrpt 10
5. Airline Fairness, A_fair 11
6. Weight Class Grouping, WCG 12
D. Model Implementation 13
E. Sequential Windowing 13
IV. Results & Analysis 16
A. FCFS 17
B. PAX Delay 18
C. Vehicle Delay 20
D. Throughput 22
E. Airline Fairness 24
F. Weight Class Grouping 26
V. Conclusion 27
References Used 28
Appendix A – Model Source Code 29
Appendix B – Work Breakdown Structure 51

List of Figures

Figure 1. Illustration of the Optimization Process 14
Figure 2. Windowing Optimization Flow Chart 14
Figure 3. Number of Hourly Arrivals Scheduled Over 24 Hours 16
Figure 4. Flight Delay Results From FCFS 17
Figure 5. Average PAX Delay Results 18
Figure 6. Average Vehicle Delay By Aircraft Type, P_delay Model 19
Figure 7. New Position From Re-Sequencing 20
Figure 8. Results For Average Vehicle Delay 21
Figure 9. Results For Hourly Average Vehicle Delay 22
Figure 10. Hourly Pax Utilization 23
Figure 11. Hourly Vehicle Utilization 24
Figure 12. Airline-Based Average PAX Delay 25
Figure 13 Delay by weight class 26

I. Executive Summary

US airports use first come, first serve (FCFS) queuing to land aircraft. Unfortunately, while this has a nice simplicity, it is unlikely the best situation for all three major stakeholders: passengers, airlines, and airports. What makes FCFS less than ideal is that each aircraft has an arrival slot, but the spacing between arrival slots is dependant upon the size of the plane that just landed and the plane that is about to land. These slot sizes are mandated by the FAA and vary from a low of 72 seconds to a maximum of 280 seconds. In the US, planes are binned into one of four size categories, Heavy (H), Boeing-757 (B757), Large (L), and Small (S).

As an example, assume that the three next planes are arriving in the following order: SHS. Under the FCFS scheme, the minimum time from when the first plane lands until the last one can land is 352 seconds, about six minutes. If we could switch the ordering to SSH, the time would be 192 seconds, about 3 minutes. Assuming all of the planes showed up at the same time, this choice of ordering saves 160 seconds, enough time to land a fourth plane of size L or larger.

This paper examines five different sequencing schemes and compares the outputs to that of a sixth scheme, FCFS, to judge improvements or degradations. The six schemes are FCFS, passenger delay minimization, vehicle (or flight) delay minimization, vehicle throughput maximization, airline fairness maximization, and weight class grouping. Each was run with data from one day at LaGuardia airport. LaGuardia was chosen because it is a fairly congested airport and also has only one runway which simplified the model building.

One of the first issues we knew that we would run into was that of dimensionality. With over 500 flights in one day, the state space of the problem is simply too large for modern computers to solve in a reasonable amount of time. However, the problem has a nice structure in that flights landing at 0800 have little to do with flights landing at 1700. We broke the data into smaller overlapping windows and ran sequential optimizations over one window at a time. With this simplification, we were able to sequence the entire day's flights within a couple of hours.

For the FCFS model we found that the average flight delay for the whole day was 4.9 minutes with standard deviation of 4.7 minutes. The average passenger delay was 4.7 minutes. Approximately 85% of all flights incurred some delay and 39% were delayed by more than five minutes.

Using the Passenger Delay model we found that we could reduce the average passenger delay to 2.7 minutes, a 43% reduction over FCFS. This occurred because we bumped many smaller flights back and allowed the larger aircraft to land sooner.

In running the Vehicle Delay model, we found that we were able to reduce the average vehicle delay to 4.3 minutes, a 9% improvement over FCFS. While this produced some gains, the average passenger delay increased slightly to 5.0 minutes.

The Vehicle Throughput model produced no particular improvement. This model tried to minimize the time of landing for the very last plane. However, as we did not allow planes to land earlier than they were scheduled to arrive, when looking at the whole day, the last plane landed as early as possible in all models. When looking at the schedule on an hour by hour basis, there were changes of only one or two flights per

hour. This is likely due to LaGuardia's homogenous mixture of planes which is 90% L, 7% B757, and 3% S.

The Airline Fairness maximization looked at the 17 airlines that had more than 200 passengers per day and attempted to equalize the average per passenger delay. For FCFS, the average per passenger delay for this subgroup was 5.0 minutes with a standard deviation of 2.0 minutes and a range of 7.4 minutes. The per passenger delay for this Airline Fairness model was worse at 5.9 minutes, but both the standard deviation and range were smaller at 1.9 minutes and 6.5 minutes, respectively.

Finally, the Weight Class Grouping model was a heuristic technique that creates groups of aircraft of the same weight class and land them in batches. The idea was to have a technique that could produce better results than FCFS but without the computational complexity of an optimization routine. The average passenger delay was 4.2 minutes with a standard deviation of 5.7 minutes. The average vehicle delay was 4.5 minutes with a standard deviation of 5.7 minutes.

While the Passenger Delay model gave the single biggest improvement, it heavily penalizes small flights to achieve those gains. Commuter flights which are scheduled for an hour in duration would often take one and a half hours, something that would not be favorable with the passengers or airlines. Additionally, these results are highly data dependant. Data from another airport with a more heterogeneous mixture of planes may yield a very different set of results. However tempting it may be, given this information, it is unwise to crown a winner.

Future work on this problem should consider implementing dynamic window sizes for the optimizations as this would likely allow large gains in performance and possibly produce better results if those given here are suboptimal. Additional analysis on other airports with more heterogeneous flight mixtures would be useful to see if any gains were to be had. Additionally, the models could be modified to handle small advances of flights, i.e. landing early.

II. Introduction

A. Background

Currently, most airports in the US land planes on a first come, first served (FCFS) basis. While this seems like a fair and easy system to use, it is not the most efficient and is vulnerable to abuse. Airlines have been known to try and game the system by filing incorrect flight plans which they know will win approval and then changing them mid-flight. By cutting corners out of their scheduled route they try to move up their arrival time despite efforts from air traffic management authority to smooth out the flow of traffic.

Additionally, in all but a few airports, airlines are free to schedule arrivals and departures at will. This leads to clusters of closely spaced flights separated by gaps during which the runway is idle. More efficient use of the runway would benefit all stakeholders including passengers, airlines, and airports.

The FAA mandates separation distances between plane landings for safety. These distances depend on the respective sizes of the plane that just landed and the one that is next in line. Based on these separations, some reordering of arrival slots by moving an airplane ahead or behind in the queue by plus or minus ten positions could allow greater

throughput of aircraft and create a more efficient use of the service resource, which is in this case the runway.

B. Problem Statement

The air traffic industry has many stakeholders, most notably passengers who fly and pay for the service. Other stakeholders are the airlines that operate the aircraft and airports that control the ground infrastructure. Each of these stakeholders plays a key role in the overall industry and each has their own perspective on what would be best.

Passengers, for example, do not want to be late. They would want minimal arrival delay from the traffic system.

Airlines want to be profitable which means attracting passengers. One way to do this is by keeping the percentage of flights delayed as low as possible. If an airport to which the airline offers service has chronic delays, the airline is negatively affected. This is especially true if the airport in question is a hub of the airline. Airlines also want to keep their own flights on time so that the compounding effect of a delayed flight on the remaining day's schedule is kept in check. For example, when a flight is delayed, it can affect the airline's gate schedule, the next outbound flight and subsequent flights by that vehicle, and other outbound flights that require the delayed crew. Thus, airlines want minimal vehicle delay (overall but especially for their own flights) and minimal passenger delay from their own flights.

Airports want to get as many flights as possible into the airport to raise money directly and drive commerce in their concourses and respective cities. Their objective is to maximize vehicle throughput and minimize vehicle delays.

Any new solution to sequencing arrivals over the current FCFS discipline has to consider ways to ensure its scheme does not favor one airline over another. The new sequencing strategy must also be fair to the stakeholders, who are passengers, airlines, and airports. The problem that we are then faced with is this: "given multiple competing interests, how can arrivals be sequenced fairly and efficiently?"

III. Modeling Approach

A. Performance Metrics

In addition to tracking individual flight delay, which is the difference between assigned and scheduled arrival times, the distribution of flight delays is of interest and is characterized by its average and standard deviation.

Average passenger delay, defined by the sum over all flights of the product of flight delay and number of passengers on that flight scaled by the total of all passengers is also reported. A modification of this metric is used for airline fairness and is defined below in the section for that model.

Vehicle or passenger throughput is defined as the number of vehicles or passengers per unit time, respectively. Capacity is defined as the maximum theoretical throughput. We have assumed a capacity for vehicles based on the minimum separation for a stream of 100% large aircraft and a capacity for passengers based on 100% large aircraft carrying 100 passengers. These capacities are 43.3 veh/hr and 4333 passengers/hr. Finally, utilization is defined as throughput scaled by capacity.

B. Assumptions

1. Deterministic Optimization

All the flights show up on their scheduled arrival time.

2. No Network effect

For the purpose of our study we assume that the airport is isolated and there is no network propagation of delays due to reshuffling.

3. Capacity Estimation

Utilization is the ratio of throughput and capacity. We estimate the capacity by assuming a case of a fleet having 100% large aircrafts. This assumption is reasonable since the given data has fleet mix: 0% heavy, 7% Boeing 757, 90% large, and 3% small. Refer to Results and Analysis Section IV for details.

4. Aircraft cannot be delayed more than 30 minutes

This assumption puts an upper cap on the amount of time an aircraft can be held up in the air before it is assigned a landing slot. It prevents aircraft from being delayed indefinitely.

5. No early arrivals allowed

For the purpose of this study we assumed that an aircraft could not arrive before its Scheduled arrival time. Even though this seems like an incoherent assumption, especially from a passenger's perspective who does not mind arriving a few minutes early, an early arrival is not desirable in common practice. An early aircraft sits and waits for a marshaller to aid with final parking, and then waits for someone to position the jet way and open the door. Because flight attendants are delayed inbound, the boarding process for the next aircraft is delayed and the aircraft departs late. This is described in the paper "The Network Airline Production Problem" by R. Baiada.

C. Models

The benchmark model for this study is a FCFS queuing discipline. This closely models the arrival sequencing strategy used in practice.

The formulation of the four optimization strategies is based on the machine scheduling model which is well known in the field of optimization. Other models were considered, but it was believed that the machine scheduling model would lend itself to the problem nicely and in a straightforward way. Decision variables and constraints in the machine scheduling model are analogous to those in our problem.

We did consider the general assignment model, and successfully applied this to the aircraft problem for the case in which arrival slot sizes were uniform (results omitted). In order to account for variations in slot sizes due to aircraft size and wake vortices, however, the machine scheduling model was utilized.

The four objective functions in our optimization models are as follows:

- minimize total passenger delay
- minimize total vehicle delay

- minimize time of the last plane (to maximize throughput)
- minimize the greatest airline specific passenger delay (to increase airline equity)

The general form of the machine scheduling model assumes there is some number of machines available to process different jobs. In some cases, only certain jobs can be performed on a given machine. For example machine X adds part A to the product whereas machine Y adds part B, before the product can be completely assembled. Accordingly, some jobs must be executed in a particular order, and the machine scheduling model can account for this. Additional constraints can also be added including a product completion time for any particular job or sequence of jobs.

The aircraft sequencing problem modeled in this study assumes that one runway is available for arrivals. This runway is analogous to one machine that processes jobs. Each flight is a job and only one flight can occupy the runway at any time. The optimization routine must assign a landing slot or start time for each job. This slot time, which is the main decision variable, must not be sooner than the aircraft is available to land. The assigned time must also be later than the previous plane's start time by at least the separation time mandated by the FAA for safety.

The main constraints in the model are designed to ensure minimum separation. They compare every pair of planes to be considered and force the absolute difference between their landing times to be at least the minimum separation time. These minimum separation times are known in advance and (as our model ignored wind effects) depend only on the size of each plane. These times are shown in Table 1. There are four categories for size in the U.S. These are, in decreasing order of size, heavy, Boeing 757, large, and small. The larger the plane, the more wake turbulence it creates behind it and the more air turbulence it can safely fly through. This makes the situation in which a small immediately follows a heavy the worst of all sequence pairs in terms of separation time (280 sec) and hence poor traffic throughput.

Table 1 Aircraft Separation Requirements

Time Separation (sec)		Trailing aircraft			
		Heavy	B757	Large	Small
Leading aircraft	Heavy	96	137	157	280
	B757	96	103	121	271
	Large	72	77	83	182
	Small	72	77	83	120

1. First Come, First Serve (FCFS)

The first queue discipline we modeled was FCFS with tie breaking based on a random variable. As mentioned previously, FCFS is the system currently used at most airports, so we used this as a baseline for the rest of our models. FCFS was relatively simple to implement and verify the solution; it had the added advantage of extremely fast solution times.

We formulated the problem as follows:

$$\forall i, j \in \{i < j\} : \{eta_i \le eta_j \quad x_j - x_i \ge sep_{i,j}\}, i, j : 1, 2, ...N \qquad (1)$$

$$\forall i, \quad x_i \ge eta_i \qquad (2)$$

There are N total flights. The earliest time of arrival for a flight, *i*, is given by eta_i. The minimum separation time for a pair of flights (given in Table 1) is represented by $sep_{i,j}$. The final landing time assigned to the flight by the optimization is x_i. Constraint 1 says that for a pair of flights [*i,j*], if they have different ETAs, the time separation between the final landing times, x_i, must be at least the minimum separation distance. If they have the same ETA, then we choose one flight at random to go first, namely flight *i*, and then assign the minimum separation accordingly. Constraint 2 ensures that the final arrival time cannot be earlier than the scheduled arrival time.

2. Passenger Delay Minimization, P_delay

Having completed the baseline case, we next looked at optimizing landing sequences with respect to our three defined stakeholders: passengers (PAX), airport, and airlines. For optimization with respect to PAX our objective was to minimize the total amount of passenger delay, defined as the sum over all flights of the product of the number of passengers on a flight multiplied by the delay (measured in seconds) for that flight. While this gets more passengers through the system more quickly, it has the drawback of penalizing smaller flights in favor of larger ones.

The formulation for this approach is:

$$MIN \quad z = \sum_i PAX_i * delay_i \qquad (3)$$

s.t.

$$\forall i, j \in \{i < j\} : \begin{Bmatrix} eta_i - eta_j \ge sep_{j,i} - My_{i,j} \\ eta_j - eta_i \ge sep_{i,j} - M(1 - y_{i,j}) \end{Bmatrix} \qquad (4)$$

$$\forall i, \quad delay_i = x_i - eta_i \qquad (5)$$

$$\forall i, \quad x_i \ge eta_i \qquad (2)$$

$$\forall i, \quad delay_i \le 1800 \qquad (6)$$

$$i, j : 1, 2, 3, ...N$$

There are N total flights. The earliest time of arrival for a flight, *i*, is given by eta_i. The minimum separation time for a pair of flights (given in Table 1) is represented by $sep_{i,j}$. The final landing time assigned to the flight by the optimization is x_i. PAX_i is the number of passengers on flight *i*, and $delay_i$ is the delay in seconds of flight *i*. We use a binary decision variable, $y_{i,j}$, to ensure that only one of the two constraints explained above in Constraint 4 is binding.

Because this is an optimization we have added Equation 3, the objective function, which says that we are trying to minimize the sum of total passenger-seconds. Passenger-seconds are calculated by multiplying the number for passengers on flight *i* by the delay in seconds for flight *i*. Constraint 4 replaces Constraint 1 from FCFS and says that if $y_{i,j}$ is 0, then the top constraint is binding and the bottom one is slack. If $y_{i,j}$ is 1, then the opposite is true. *M* is a large constant which ensures that the right hand side of the inequality is always less than the left hand side. Again, we use Constraint 2 to ensure the final arrival time cannot be earlier than the scheduled arrival time. Finally, we say that the delay (defined by Equation 5) for any individual flight cannot be more than 30 minutes (1800 seconds) in Constraint 6.

3. Vehicle Delay Minimization, V_delay

To optimize the timely arrival of vehicles, we used an objective that minimized the sum of delay for all vehicles. Although similar to the PAX optimization discussed above, this treats every plane of equivalent size, regardless of passengers, as equal.

The formulation for this approach is:

$$MIN \quad z = \sum_i delay_i \tag{7}$$

s.t.

$$\forall i, j \in \{i < j\}: \begin{Bmatrix} eta_i - eta_j \geq sep_{j,i} - My_{i,j} \\ eta_j - eta_i \geq sep_{i,j} - M(1 - y_{i,j}) \end{Bmatrix} \tag{4}$$

$$\forall i, \quad delay_i = x_i - eta_i \tag{5}$$

$$\forall i, \quad x_i \geq eta_i \tag{2}$$

$$\forall i, \quad delay_i \leq 1800 \tag{6}$$

$$i, j : 1, 2, 3, ...N$$

There are N total flights. The earliest time of arrival for a flight, *i*, is given by eta_i. The minimum separation time for a pair of flights (given in Table 1) is represented by $sep_{i,j}$. The final landing time assigned to the flight by the optimization is x_i. The delay in seconds of flight *i* is $delay_i$. We use a binary decision variable, $y_{i,j}$, to ensure that only one of the two constraints in Constraint 4 is binding.

This formulation is almost identical to the Passenger Delay Minimization. All of the constraints (2, 4, 5, and 6) have carried over and operate identically as described above. The change here is only in the objective function, Equation 7, which is now minimizing the sum of the delays for each flight, unweighted by the number of passengers on each flight. Constraint 4 says that if $y_{i,j}$ is 0, then the top constraint is binding and the bottom one is slack. If $y_{i,j}$ is 1, then the opposite is true. *M* is a large

constant which ensures that the right hand side of the inequality is always less than the left hand side. Again, we use Constraint 2 to ensure the final arrival time cannot be earlier than the scheduled arrival time. Finally, we say that the delay (defined by Equation 5) for any individual flight cannot be more than 30 minutes (1800 seconds) in Constraint 6.

4. Vehicle Throughput Maximization, V_ thrpt

For airport optimization, the objective was to complete the given list of arrivals within the shortest amount of time. This is analogous to generating the maximum throughput of vehicles. This optimization also treats each plane equally regardless of size, however, it can increase delays for some flights in order to increase vehicle throughput.

We used the following formulation:

$$MIN \quad z = time_{lastplane} \tag{8}$$

$$s.t.$$

$$\forall i, j \in \{i < j\}: \left\{ \begin{array}{c} eta_i - eta_j \geq sep_{j,i} - My_{i,j} \\ eta_j - eta_i \geq sep_{i,j} - M(1 - y_{i,j}) \end{array} \right\} \tag{4}$$

$$\forall i, \quad delay_i = x_i - eta_i \tag{5}$$

$$\forall i, \quad x_i \geq eta_i \tag{2}$$

$$\forall i, \quad delay_i \leq 1800 \tag{6}$$

$$\forall i, \quad time_{lastplane} \geq x_i \tag{9}$$

$$i, j : 1, 2, 3, \ldots N$$

There are N total flights. The earliest time of arrival for a flight, *i*, is given by eta_i. The minimum separation time for a pair of flights (given in Table 1) is represented by $sep_{i,j}$. The final landing time assigned to the flight by the optimization is x_i. The landing time assigned to the last plane is captured by $time_{lastplane}$. The delay in seconds of flight *i* is $delay_i$. We use a binary decision variable, $y_{i,j}$, to ensure that only one of the two constraints in Constraint 4 is binding.

This formulation is almost identical to the previous two. All of the constraints (2, 4, 5, and 6) have carried over and operate identically as described above. There are two changes here: the objective function, Equation 8, which is now minimizing the time that the last plane lands, and Constraint 9 which sets $time_{lastplane}$ to the time of the latest flight. Constraint 4 says that if $y_{i,j}$ is 0, then the top constraint is binding and the bottom one is slack. If $y_{i,j}$ is 1, then the opposite is true. *M* is a large constant which ensures that the right hand side of the inequality is always less than the left hand side. Again, we use Constraint 2 to ensure the final arrival time cannot be earlier than the scheduled arrival time. Finally, we say that the delay (defined by Equation 5) for any individual flight cannot be more than 30 minutes (1800 seconds) in Constraint 6.

5. Airline Fairness, A_fair

For airline fairness optimization, the objective was to consider all groups of flights arriving at the same time and to then allocate PAX delay evenly over the airlines in competition for arrival slots. FCFS is used to roughly sequence all flights, and optimization is used to finely sequence the subgroups of two to six flights having the same pre-scheduled arrival times.

We limited the optimization to airlines that had more than 200 passengers per day. This was primarily to eliminate general aviation flights which may not be constant from day to day and so would not appear in a standard schedule. Reducing the number of airlines considered also has a huge improvement on runtime because it reduced the dimensionality of the problem.

This optimization differs from the others that we are presenting in that rather than using the data as a surrogate for a live data feed, it can be thought of as a way of generating a daily schedule that is fair from the start. This hypothetical schedule is based on airlines' preferences for landing slots.

We used the following formulation:

$$MIN \quad z = penalty_{\max} \tag{10}$$

s.t.

$$\{i<j\}: \begin{cases} eta_i < eta_j & x_j - x_i \geq sep_{i,j} \\ eta_i = eta_j & \left\{ \begin{array}{c} x_i - x_j \geq sep_{i,j} - My_{i,j} \\ x_j - x_i \geq sep_{j,i} - M(1-y_{i,j}) \end{array} \right\} \end{cases} \tag{11}$$

$$\forall i, \quad x_i \geq eta_i \tag{2}$$

$$\forall i, \quad delay_i = x_i - eta_i \tag{5}$$

$$\forall i, \quad delay_i \leq 1800 \tag{6}$$

$$\forall k, \quad penalty_k = \frac{\sum_{i \in S_k} PAX_i * delay_i}{\sum_{i \in S_k} PAX_i} \tag{12}$$

$$\forall k, \quad penalty_{\max} \geq penalty_k \tag{13}$$

$i, j: 1, 2, 3, ...N$

k: *index over airlines*

There are N total flights. The earliest time of arrival for a flight, *i,* is given by eta_i. The minimum separation time for a pair of flights (given in Table 1) is represented by $sep_{i,j}$. The final landing time assigned to the flight by the optimization is x_i. PAX_i is the number of passengers on flight *i*, and $delay_i$ is the delay in seconds of flight *i*. The penalty for a give airline *k*, is $penalty_k$ and the largest of those is $penalty_{max}$. S_k is the set of all flights in contentious slots operated by airline *k*. We use a binary decision variable, $y_{i,j}$, to ensure that only one of the two constraints in Constraint 4 is binding.

This formulation might be better considered as an unfairness minimization. The constraints (2, 5, and 6) have carried over and operate identically as described above.

Once again, we have a new objective function, Equation 10, which minimizes the maximum penalty assigned to a given airline. Constraint Set 11 replaces Constraint Set 4. Since this optimization is based around the FCFS methodology, we have the first line which says that if $flight_i$ is scheduled to land before $flight_j$, then ensure the minimum separation time. If the scheduled arrival times are the same, then we need to choose an ordering and ensure a minimum separation distance. The other constraint becomes nonbinding and we enforce this by setting $y_{i,j}$ to either 0 or 1. Each $penalty_k$ is calculated in Equation 12; it is the total delay for passengers on flights in contentious slots for an airline divided by the total number of passengers on flights in contentious slots for an airline. This could also be referred to as the average passenger delay expressed in seconds per passenger. Constraint 13 assigns $penalty_{max}$ the largest value of all $penalty_k$'s. Constraint 4 says that if $y_{i,j}$ is 0, then the top constraint is binding and the bottom one is slack. If $y_{i,j}$ is 1, then the opposite is true. M is a large constant which ensures that the right hand side of the inequality is always less than the left hand side. Again, we use Constraint 2 to ensure the final arrival time cannot be earlier than the scheduled arrival time. Finally, we say that the delay (defined by Equation 5) for any individual flight cannot be more than 30 minutes (1800 seconds) in Constraint 6.

The raw data required preprocessing to create a two-column matrix in order to allow the MPL program to calculate $penalty_k$ (see Appendix A.5).

6. Weight Class Grouping, WCG

While FCFS is easy to implement and understand, we know the results are generally suboptimal. The results from our optimization models tell us the best possible solution for their respective objectives, but it comes at the cost of huge computation times. To try and find a balance between speed and optimality, we developed a Weight Class Grouping model in Java.

Table 1 shows that the required separation for a larger aircraft followed by a smaller aircraft (e.g. H-B757) is longer than the separation needed for a same weight class pair (e.g. H-H). Therefore, it would be desirable to group arrivals into batches of aircraft in the same weight class. In this model, we accomplish this task with the following heuristics:

- The first scheduled arrival flight is set to be the first arrival. Then, based on the following heuristics, we search and determine the second arrival. After that, we search and determine the third arrival, and so forth, until all flights are resequenced.
- If the scheduled separation between $flight_i$ and $flight_{i+j,}$ $(j\geq1)$ is greater than that mandated by the FAA (i.e. separation standard constraints are nonbinding), arrivals are sequenced based on FCFS.
- If there is a j, $(j\geq1)$ such that the scheduled separation between $flight_i$ and $flight_{i+j,}$ is less than that mandated by the FAA (i.e. separation standard constraints are binding), all arrivals which make the separation standard constraints binding are checked and compared. Preference for the next arrival is given to aircraft in the same weight class as the preceding arrival. If no aircraft of the same weight class is available, we start landing a batch of aircraft from another weight class.
- When changing the weight class batch, we start by moving from smaller weight classes to larger ones. That is, if we just landed a batch of Small aircraft we move

to Heavy, then Heavy to B757, and B757 to Large. After the heaviest weight class batch is landed, we change directions and start descending in weight class size. Once we reach the smallest weight class with flights, we again switch directions and continue in the fashion until all flights have been resequenced.

- If separation standard constraints for a certain pair of aircraft become nonbinding, when arrivals sequence is based on FCFS, the direction is set to be ascending the next time the separation constraints become binding.

As an example, the sequence H-S requires the largest separation, 280 seconds. If we can swap the order to S-H the separation is only 72 seconds, a savings of 208 seconds. If we cannot swap the pair and the next flight is L, then we try swapping S with that flight. Over the three flight sequence, this could save up to 24 seconds. While 24 seconds isn't much on its own, if we can make a few of these swaps each hour, we can fit in one or two more flights per hour.

D. Model Implementation

For the optimization portion of our solution we used MPL to encode our mathematical models (P_delay, V_delay, V_thrpt, and A_fair). MPL is a macro language and development environment designed to automate the creation of mixed integer linear programs (MILP). MPL is also integrated with CPLEX which is currently the industry standard MILP solving engine. This means that once we have described our method in MPL, CPLEX begins to work on solving the problem. There are other competing products with MPL such as OPL and AMPL; however, MPL was chosen because each member of the team had used it in previous coursework, and it was therefore the product with the smallest learning curve.

E. Sequential Windowing

Trying to optimize all flights scheduled to land at an airport in a given day at once is not realistic on stand alone computers today. However, we can exploit the structure of the problem a bit to try and come close to an optimal scheduling. Flights landing at 8:00 a.m. have little to do with those landing at 5:00 p.m. Using flight arrival data from the first of June 2006 for La Guardia airport as a surrogate for an actual live feed, we pull a fixed window of n flights, optimize their sequence, and then fix the arrival of the first one. We then add the next flight (flight n+1 from the original list) to the window and rerun the optimization. We continue to iterate in this way until we have sequenced the entire day's worth of flights. See Figure 1 for an illustration of the optimization process.

To automate this task we used the Optimax2000 C++ library provided by Maximal Software. This allowed us to read in the text file of flight data and then write the flights in that window to an excel file. We then call MPL/CPLEX through the Optimax2000 library and the optimization begins. Once it is finished we parse the results and write them back out, including any current delays introduced by the latest round of optimization. Keeping track of the delays from iteration to iteration ensures that we do not delay a flight indefinitely and that we calculate the appropriate penalties in the end. This process is explained in more detail below. Note that this process was not needed to solve for the FCFS model; furthermore, it was not suitable to solve the A_fair model, which needs to consider the entire day's worth of flights to create equity.

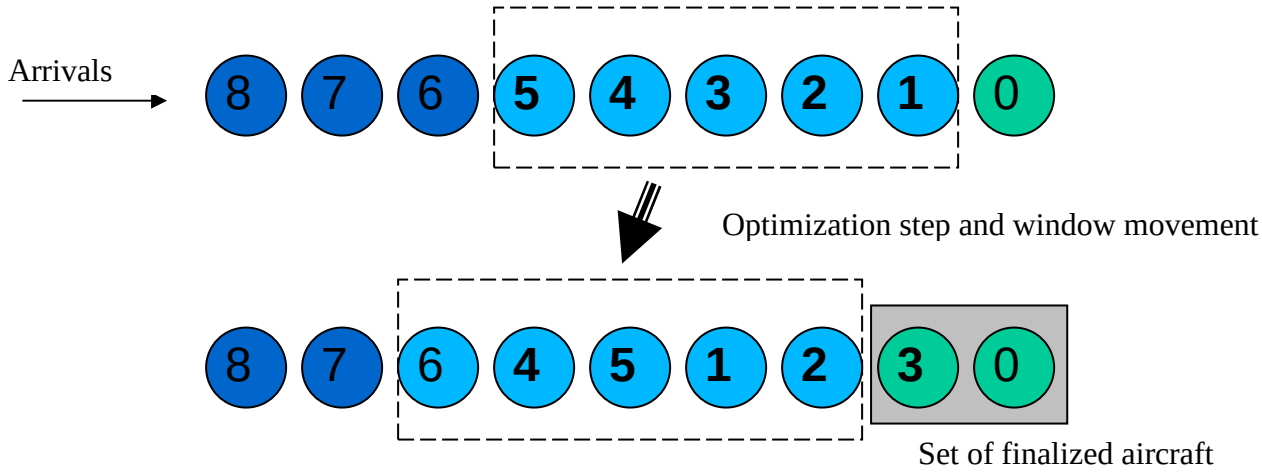

Figure 1. Illustration of the Optimization Process

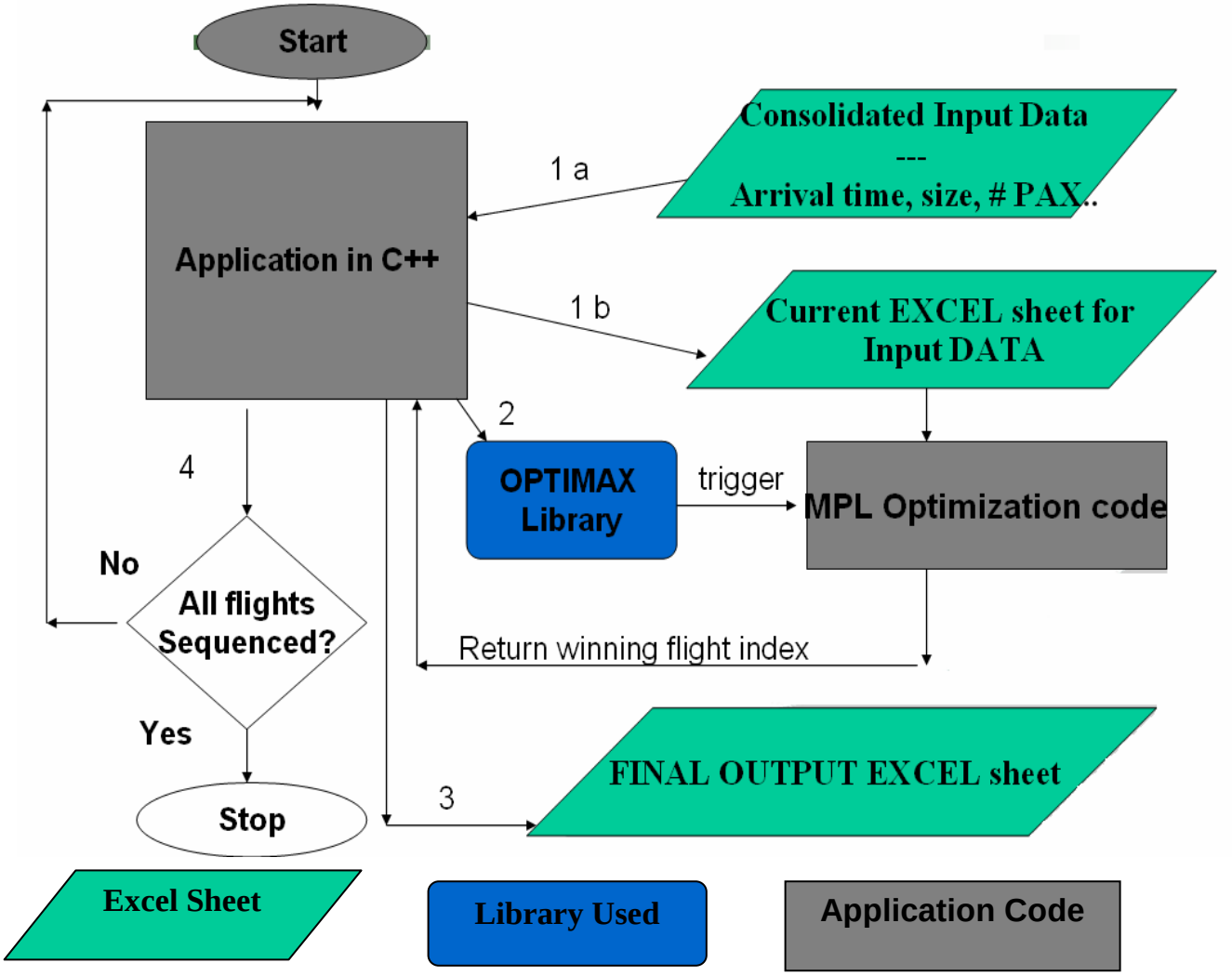

Figure 2. Windowing Optimization Flow Chart

Walk through of the process

Given a consolidated input data excel sheet, which contains all the N flights that needs to be sequenced sorted according to the scheduled arrival time. Let us denote the window size by 'w'. The first (w+1) rows from the consolidated input file is read and copied into a temporary input file (which will be used by the Optimization code in the next step). The first flight out of these (w+1) flights is already fixed. Then we reshuffle the remaining 'w' flights according to the desired strategy. The winning flight is assigned a final slot and is made the leading flight for the next iteration. This process continues until we exhaust all the N flights that needs to be sequenced.

The algorithm depicted in Figure 2 works as follows:
Given,

- a set of **N** flights to resequence. These N flights should be sorted by their scheduled arrival time.
- Let the window size be **w** (user-settable parameter)
- Let **count** denote the iteration #.
- Let $\mathbf{v_{count}}$ denote the index of the first flight for iteration #.

Do the following:

1. Pick the first 'w+1' entries of the N flights.
2. Set, maxCount = N-w & count=1
3. Set $v_1 = 1$, i.e fix the first flight on the list, i.e. make it the leading flight.
4. Reshuffle the remaining sequence of 'w' flights based on the desired optimization strategy, i.e. Min Passenger delay, Max Vehicle Throughput etc. Set '$v_{count+1}$' to the index of the first flight.
5. if (count<=maxCount)
   then, finalize the slot for flight '$v_{count+1}$',
   Make it the leading flight for the next iteration, and
   Replace the leading flight of this iteration, i.e. v_{count} by the next flight, i.e. the $(w+1+count)^{th}$ flight.
   count = count+1
   Go to Step 4
   Else, (* this is for the last window*)
   finalize the slots for all the 'w' flights in the window in the order determined by Optimization
   STOP

The two librarys used in this process were Optimax 2000 and Basic Excel- A Class to Read and Write to Microsoft Excel. Optimax is an object oriented component library used to embed an optimization into an end-user application. We used it to integrate our MPL models into the Visual C++ code. More information about the Optimax library can be found at the following URL http://www.maximal-usa.com/optimax/.

Basic Excel is an open source library to read, create and modify excel files through C++. It is fairly simple to use and documentation along with the source is available at http://www.codeproject.com/useritems/BasicExcel.asp. The MPL, C++, and Java source code for each of these models is shown in the appendix at the end of this document.

IV. Results & Analysis

LaGuardia is known to be a highly congested airport. The schedule of arrivals at LaGuardia on the day studied is quite full and consists of 523 flights in a 24 hour period. Figure 3 shows the hourly schedule of arrivals. Between the hours of 8 AM and 6 PM the hourly traffic intensity, defined as the number of arrivals per hour divided by the average service rate, is at least 0.78 (assuming the average service rate is that for large-sized craft equal to 43.3 per hour). It is safe to assume this average service rate, because the fraction of large-sized planes is high. The distribution of sizes is as follows: 0% heavy, 7% Boeing 757, 90% large, and 3% small. We demonstrate later how this high traffic intensity and particular homogeneous size distribution does not allow significant gains by re-sequencing vehicles.

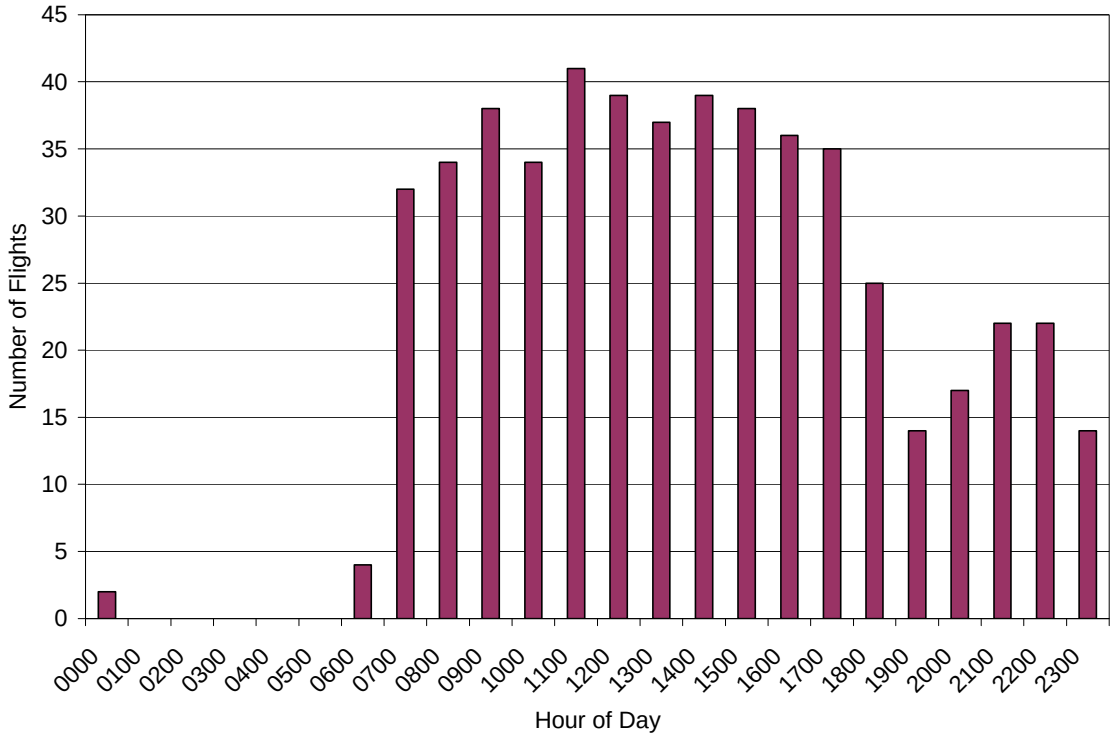

Figure 3. Number of Hourly Arrivals Scheduled Over 24 Hours

A. FCFS

Figure 4 shows significant delays from 1200 to 1800 for the FCFS discipline. The average flight delay for the whole day was 4.9 minutes with standard deviation of 4.7 minutes. The average passenger delay was 4.7 minutes. About 85% of all flights incurred some delay and 39% were delayed by more than five minutes. Passenger throughput was 2091 pax/hr (based on capacity defined by the large category, and a midrange value of 100 pax/veh) and pax utilization was 0.48. Vehicle throughput was 22.1 units/hr and corresponding utilization at 0.51.

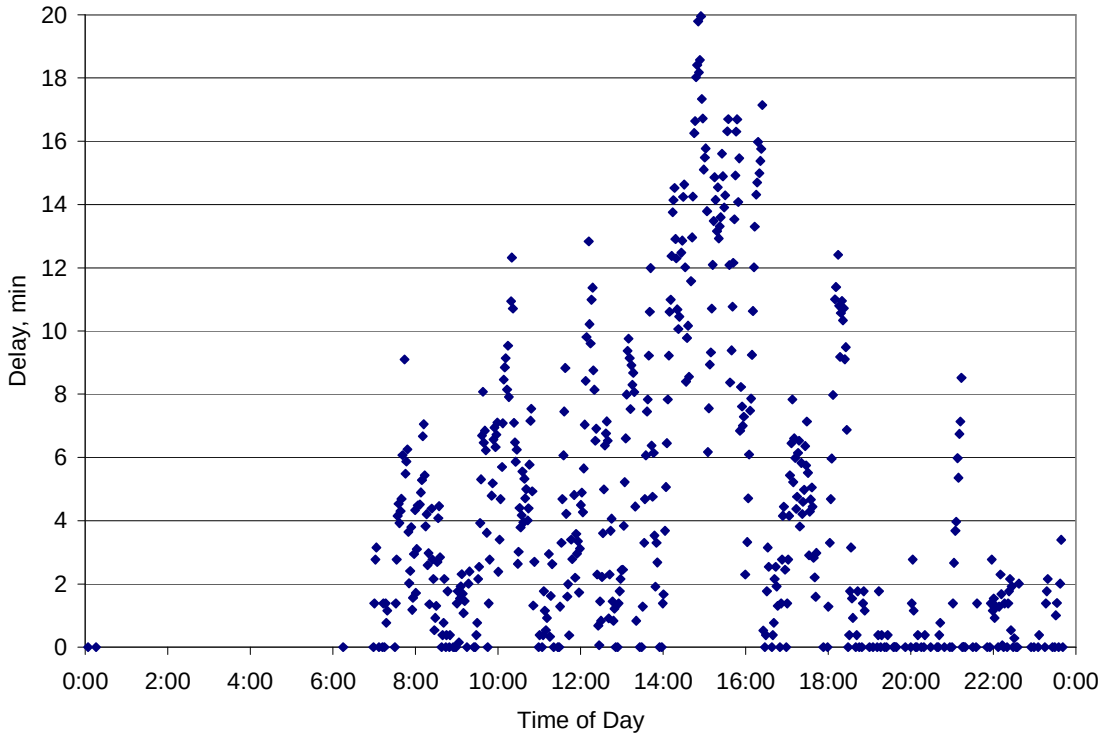

Figure 4. Flight Delay Results From FCFS

B. PAX Delay

PAX delay, shown in Figure 5, is the lowest for the P_delay model at 2.7 min/person. PAX delay is the highest for vehicle throughput optimization at 5.2 min/person. Other strategies produce PAX delay ranging from 4.6 to 5.1 min/person.

We expected P_delay to be the best strategy for minimizing pax delay, since its objective function is that very expression. Although V_delay would also provide low PAX delay results, as it did as the second best model, that model strives to minimize the sum of vehicle delay independent of the PAX per flight.

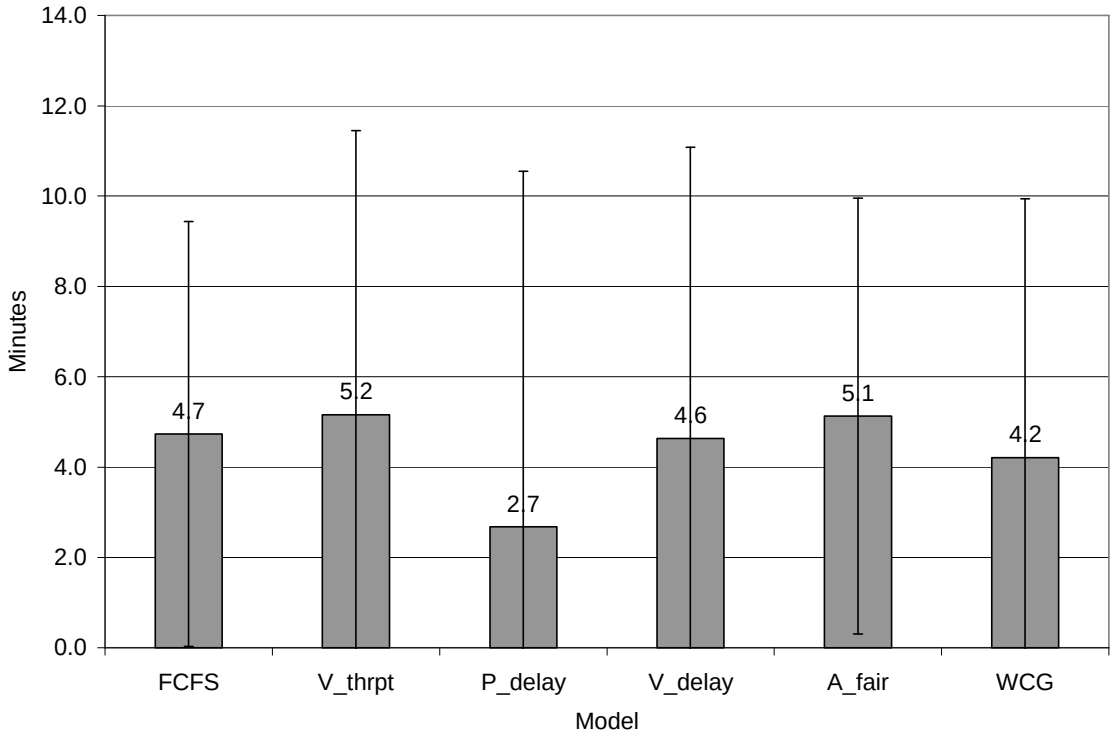

Figure 5. Average PAX Delay Results

There is a significant reduction in PAX delay when the P_delay is executed using from five to ten flights per optimization batch in the sliding window. When the window size is five flights, delay is 3.1 min/person. When the window is ten, the delay is 2.7 min/person. This indicates that the larger window allows a sequence that is closer to optimality, defined by the case in which all 523 flights could be optimized at once without needing the sliding window. Based on this result, we present results for all windowing models with a window size of ten.

To probe deeper into the P_delay model and investigate how PAX delay is significantly better, we show average vehicle delay by aircraft size in Figure 6. Clearly,

the larger the plane and corresponding PAX capacity, the lower the vehicle delay. Higher landing priority is given to larger craft.

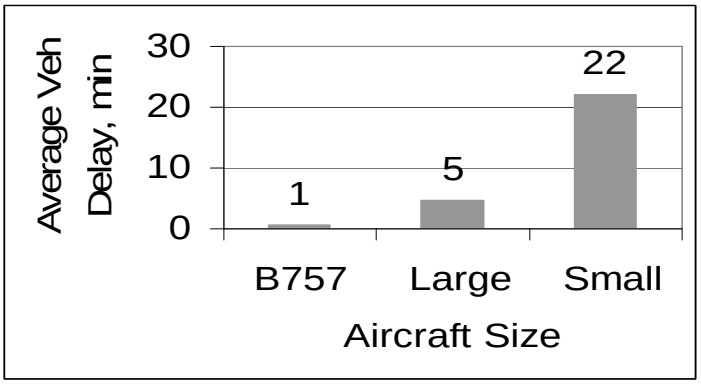

Figure 6. Average Vehicle Delay By Aircraft Type, P_delay Model

We also observe that the re-sequencing of flights can be more extensive for P_delay compared to that of FCFS. Let us define the change in position of a given flight due to re-sequencing as the absolute difference between its original position and its new position (e.g. if original position is 10 and the new position is 13, then the change is 3) and call it the reposition quantity. Figure 7 graphically compares these P_delay and FCFS reposition results. FCFS results in a maximum reposition of 6 with an average of 0.5 and standard deviation of 1.0 (note that the reposition quantity is 6 when seven flights have the same scheduled arrival time or original position and must be given unique landing times or new positions). P_delay has a maximum reposition of 19 (average of 3.3 and standard deviation of 3.6). This strengthens the argument that improvement in system performance metrics, PAX delay for example, requires more significant re-sequencing than that employed by FCFS.

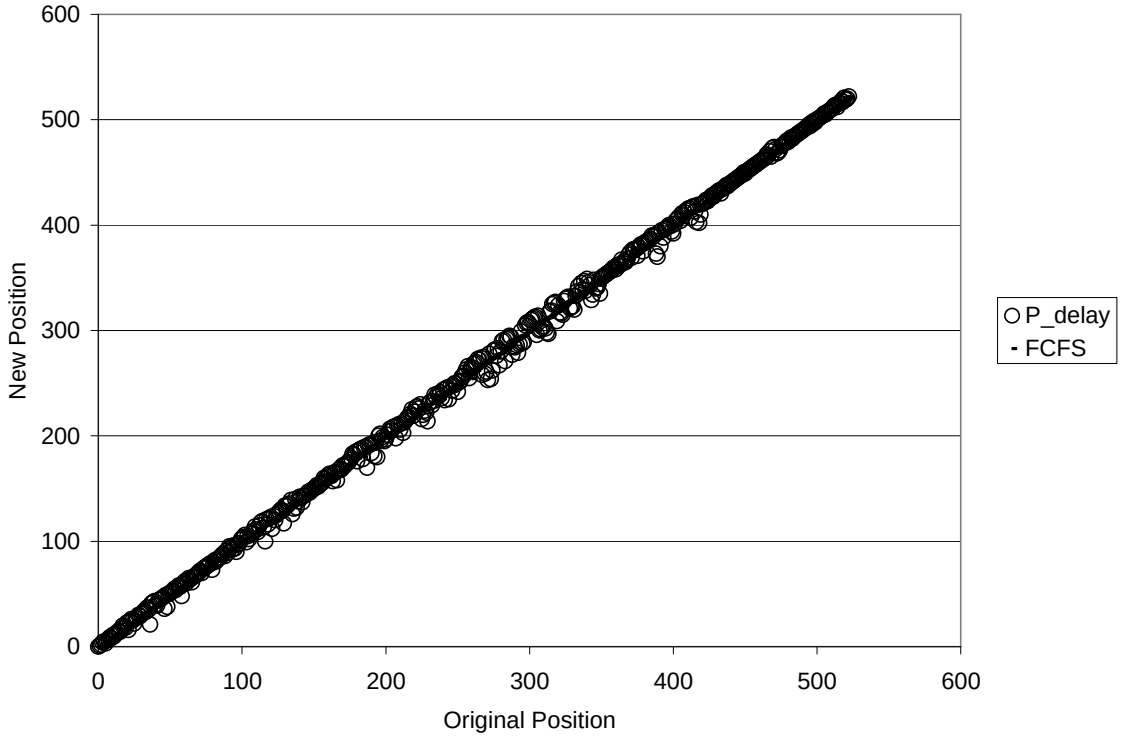

Figure 7. New Position From Re-Sequencing

C. Vehicle Delay

Not surprisingly, the best strategy for vehicle delay is the model that optimizes for this very metric resulting in a value of 4.3 min. The next best model is the vehicle throughput model whose delay metric is 4.7 min. Other models produce delay values from 5.0 to 5.2 min on average. These results are provided in Figure 8. P_delay does not yield a low average vehicle delay, because it penalizes small planes, as discussed in the previous section.

As shown in Table 2, the standard deviation of vehicle delay is lowest for the FCFS (4.7 min) and for airline fairness (which also employs FCFS as the base sequencing rule). The PAX delay model produces the highest variability in flight delay as indicated by a standard deviation of 7.9 minutes.

Also from Table 2, we see that the amount of repositioning is similar for FCFS and A_fair. This is not surprising since A_fair utilizes FCFS for course sequencing. , the amount of repositioning is similar among V_delay, P_delay, and V_thrpt.

When we compare average vehicle delay for FCFS and V_delay on an hourly basis as shown in Figure 9, we see that V_delay is a better overall model for this performance metric.

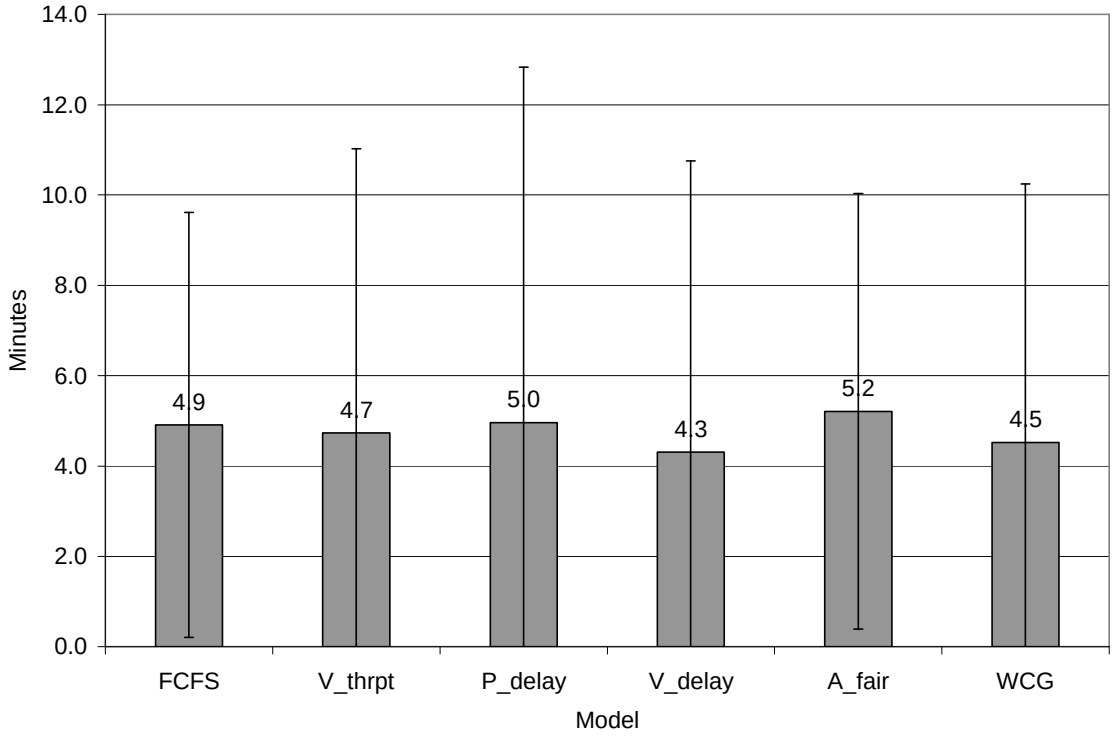

Figure 8. Results For Average Vehicle Delay

Table 2. Summary of Results for Delay Metrics

Model	Ave Pax Delay	Ave Vehicle Delay	Std Dev Vehicle Delay	Max Reposition Qty	Ave Reposition Qty	Standard Deviation of Reposition Qty
FCFS	4.73	4.91	4.70	6	0.5	1.0
V_thrpt	5.16	4.73	6.29	18	2.6	3.0
P_delay	2.68	4.96	7.87	19	3.3	3.6
V_delay	4.64	4.31	6.45	18	2.3	3.1
A_fair	5.13	5.21	4.82	5	0.4	0.8
WCG	4.21	4.52	5.73	36	1.2	2.9

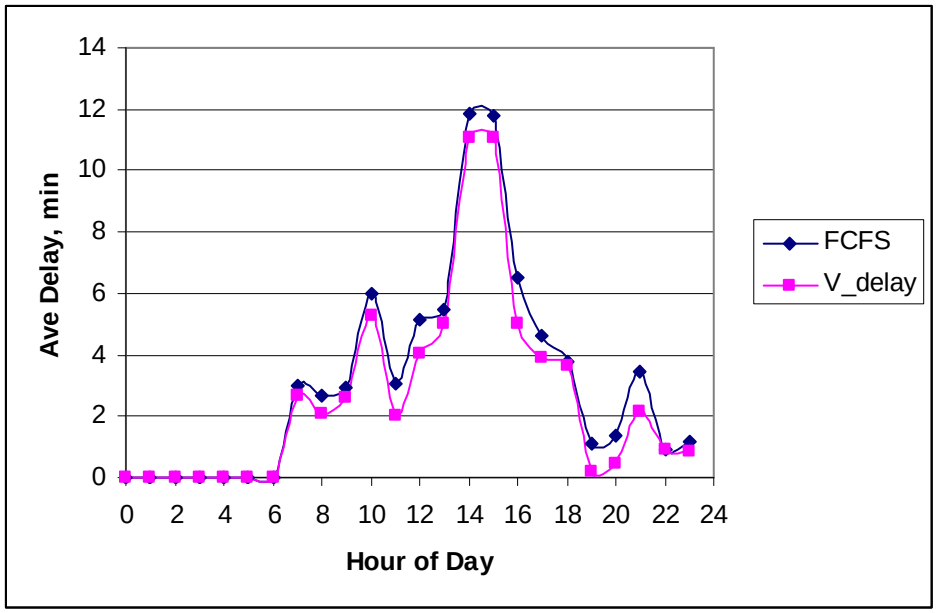

Figure 9. Results For Hourly Average Vehicle Delay

D. Throughput

The passenger and vehicle throughput values for the entire day are each constant over all models. This is because each model assigns the same landing times to the first and last planes. Accordingly, the utilization values for the entire day are also constant. See Table 3 for these and a summary of all results.

Table 3. Summary of Results

Model	Ave PAX Delay, min	Ave Vehicle Delay,min	Std Dev Vehicle Delay, min	PAX Throughput, PAX/hr	Vehicle Throughput, veh/hr	PAX Utilization	Vehicle Utilization
FCFS	4.73	4.91	4.70	2091	22.1	0.483	0.511
V_thrpt	5.16	4.73	6.29	2091	22.1	0.483	0.511
P_delay	2.68	4.96	7.87	2091	22.1	0.483	0.511
V_delay	4.64	4.31	6.45	2091	22.1	0.483	0.511
A_fair	5.13	5.21	4.82	2091	22.1	0.483	0.511
WCG	4.21	4.52	5.73	2091	22.1	0.483	0.511

When examined on an hourly basis, there is only a variation in the order of one to two flights in any hour across all models. Hourly PAX utilization is shown in Figure 10 and hourly vehicle utilization is shown in Figure 11. Note that results for all five models are quite similar. This can be attributed to several factors. Namely these are the restriction that no flights can arrive early, the high traffic intensity from 8 AM to 6 PM, the uniformity of the fleet size (90% large), and the size of the optimization window (limited to ten by computational complexity). Allowing flights to arrive early might redistribute the hourly throughput by reducing some congestion and thereby increasing throughput immediately prior to congested times.

A change in fleet size distribution from 100% small to 100% heavy, for example, would result in a higher overall or hourly throughput for vehicles and PAX. However, when utilization is less than unity (which implies that there are times when the server or runway is idle), interchanging the position of two vehicles, large and small for example, would have relatively less of an impact on throughput.

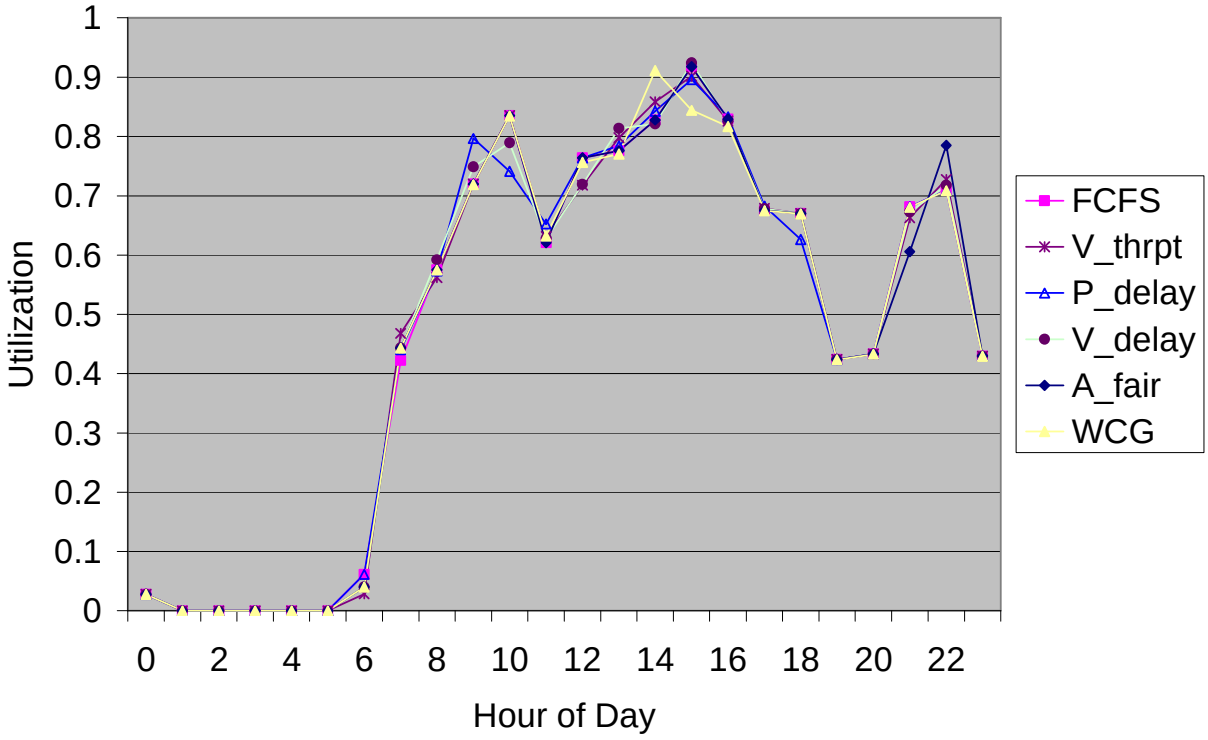

Figure 10. Hourly Pax Utilization

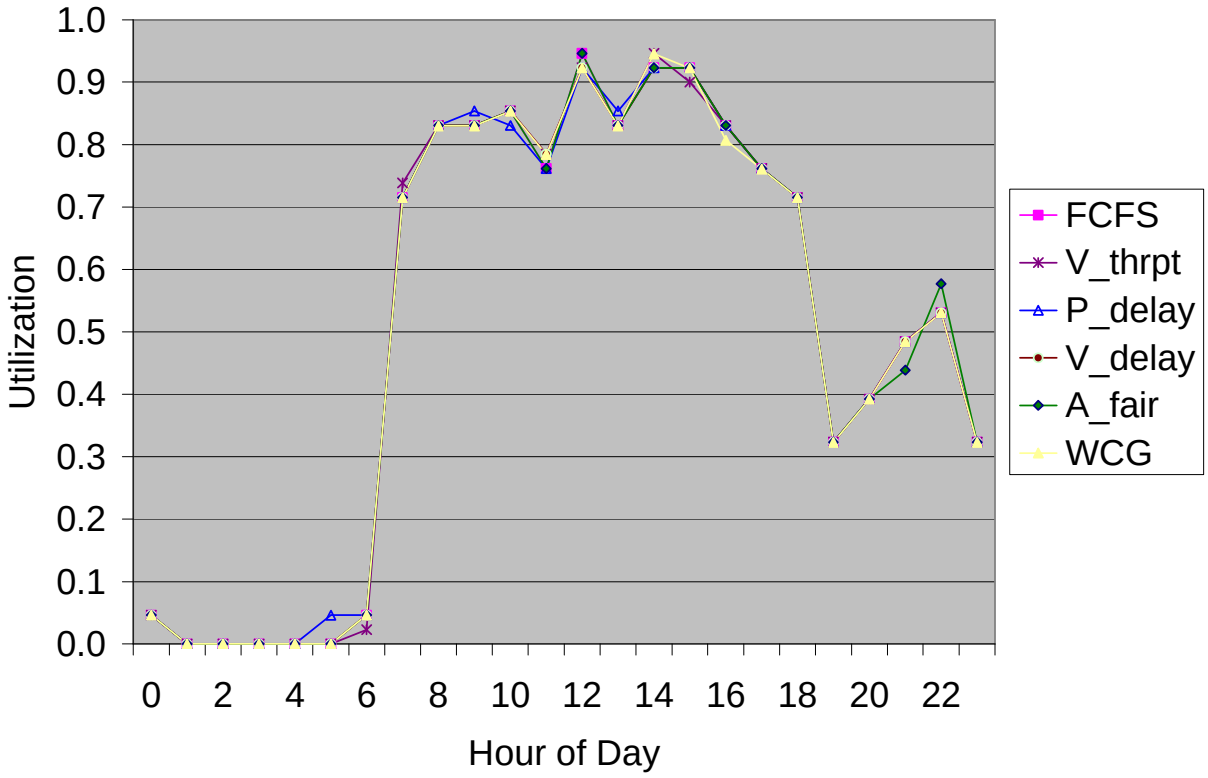

Figure 11. Hourly Vehicle Utilization

E. Airline Fairness

Since the A_fair model is essentially FCFS with selective breaking of ties, the results are similar to the benchmark FCFS model. However, in an attempt to create equity among airlines, the fairness model produces slightly higher average delays. For the following discussion, we consider only the flights that have an arrival conflict and which belong to the 17 airlines having more than 200 PAX/day (see Table 4). Table 4 shows the average PAX delay results for these airlines from the two models mentioned. These results are also plotted in Figure 12 as a function of the total PAX per airline (again only for the set of flights that have an arrival conflict).

The FCFS model results in a distribution of airline specific PAX delay values that has an average of 5.0, standard deviation of 2.0, and range of 7.4 min/person. The A_fair model's delay distribution is shifted slightly higher with average of 5.9 min/person. However, its distribution over airlines is less broad with standard deviation of 1.9 and range of 6.5 min/person (see Table 5).

Table 4. Airline-Based Average PAX Delay from Flights In Slot Contention

Airline	FCFS	A_fair
American Airlines	4.8	4.3
Air Canada	4.5	5.1
American Trans Air	2.5	1.5
Chautauqua Airlines	4.3	4.2
Colgan Air	5.9	5.5
Continental Airlines	3.9	5.0
Comair	4.9	5.5
Delta Airlines	5.4	5.5
American Eagle	7.4	7.2
Jet Blue	4.0	7.8
Midwest Exp	1.2	2.8
Spirit Airlines	4.4	7.5
Northwest Airlines	2.8	6.7
Usair Exp	7.9	7.8
Airtran Airlines	7.4	8.0
United Airlines	8.6	8.0
Usair	5.7	6.9

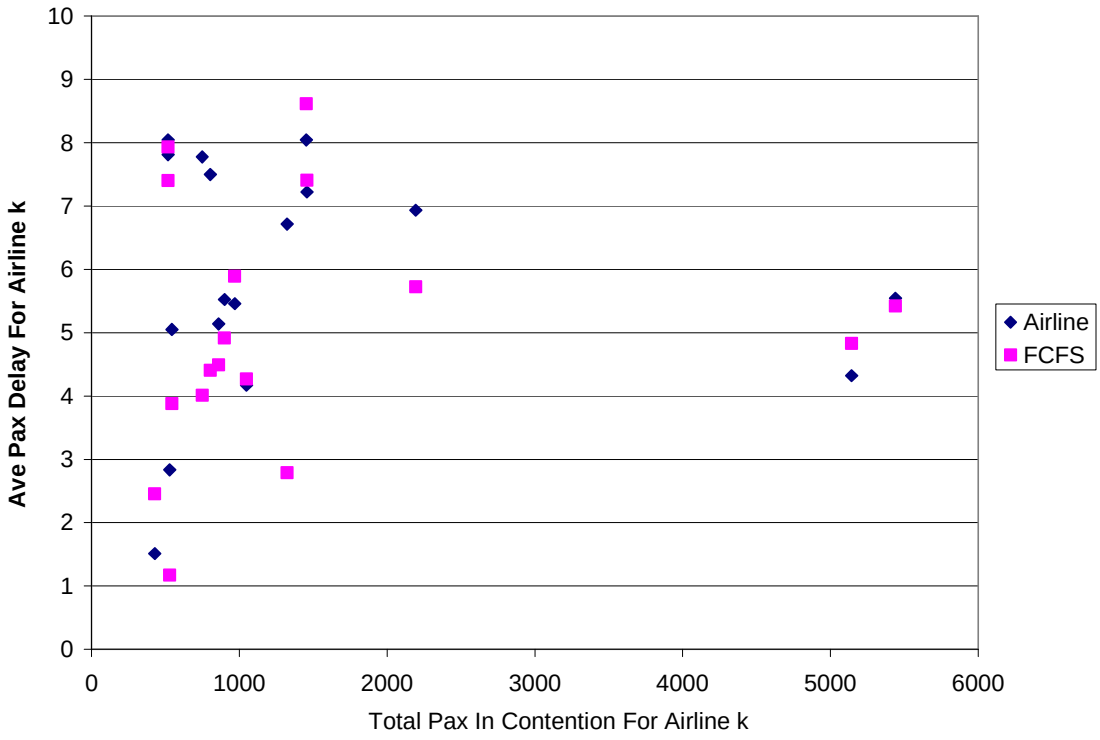

Figure 12. Airline-Based Average PAX Delay

Table 5. Summary of Airline PAX Delay Distribution

PAX Delay (Function of Airline k)	FCFS	A_fair
Average	5.0	5.9
Standard Deviation	2.0	1.9
Range	7.4	6.5

F. Weight Class Grouping

The weight class grouping strategy tries to group same weight class aircrafts together. Since the dominant weight class in LGA is Large, Large aircraft are usually grouped while B757 and Small aircraft are pushed back which produces high delay for those types, as shown in Figure 13.

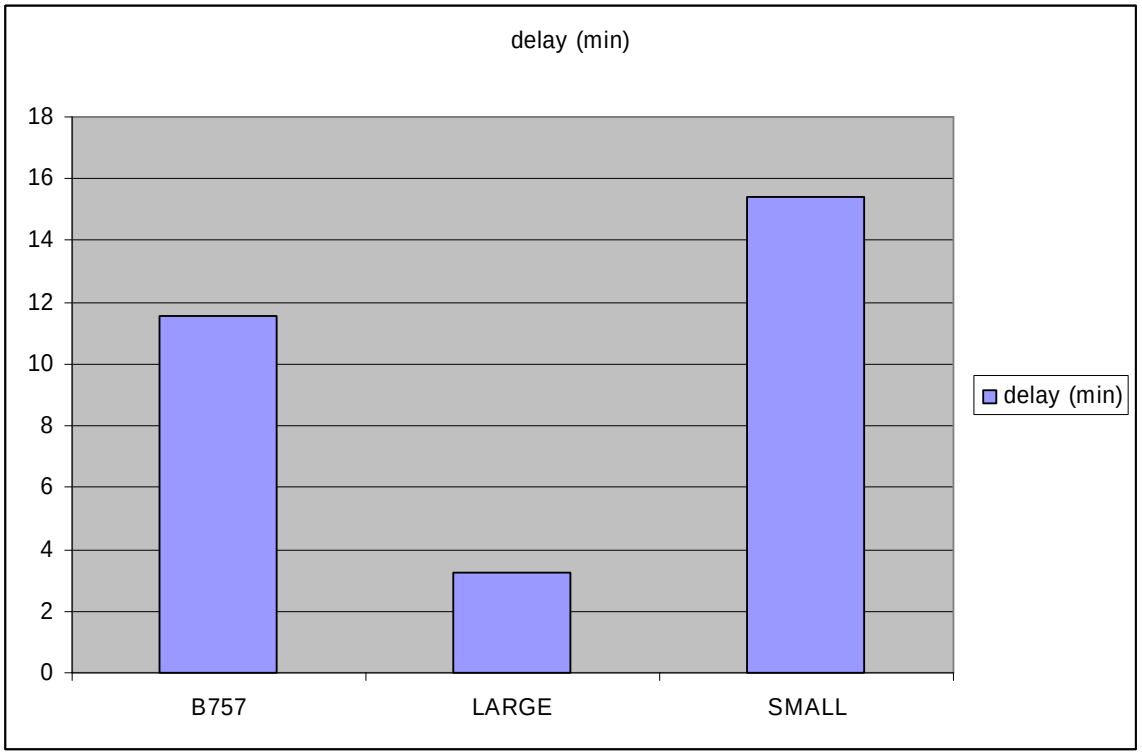

Figure 13 Average delay by weight class

V. Conclusion

Overall we saw some minor advantages over the current FCFS system. This is likely due to the airport having a utilization of over 0.8 for 10 of the 18 hours that it was operating. This high utilization (and resulting congestion) of LaGuardia makes it difficult to find opportunities for improvement. The other issue is the largely homogeneous mixture of aircraft type, 90% large, 7% Boeing 757, and 3% small. With that many aircraft of a single type, small localized reorderings are not going to find improvements by closing gaps in the arrival slots.

In the one case where we saw noticeable improvement, PAX Delay minimization, the improvement came from bumping the small flights for as long as possible. It is hard to imagine that this scheme would be able to have buy-in from airlines that primarily operate smaller regional jets or general aviation flights. However, it is important to note that such a small fraction (3%) of this particular size aircraft so greatly disrupts the on-time arrival of other larger flights. Specifically, if the average PAX delay is reduced from 4.7 for FCFS to 2.7 min/psn for PAX delay minimization by unfairly delaying small planes, then one might be led to inquire about the effect of removing such flights entirely.

While it is tempting to crown a winner from this group of algorithms, it cannot be stressed enough that we only looked at one day's worth of data at one airport and we only considered the landing schedule. We made no attempts to model gate constraints or when a plane would be next required for service.

Future work could include expanding these models to handle airports with multiple landing strips. This would allow use of data from many more airports. Additionally, finding an airport with a more heterogeneous traffic mixture might see more improvement.

References Used

Baiada, R. Michael. *The Network Airline Production Problem*. ATH Group. June 2005.

Donohue, G. L. *Air Transportation: A Tale of Prisoners, Sheep and Autocrats.* GMU Vision Lecture Series. 2007.

Harikiopoulo, D., Neogi, N. *Polynomial Time Feasibility Condition for Multi-Class Aircraft Sequencing on a Singe Runway Airport*. AIAA 1st Intelligent Systems Technical Conference. 2004.

Neuman, F., Erzberger, H. *Analysis of Sequencing and Scheduling Methods for Arrival Traffic*, NASA Technical Memorandum.1990.

Yen , J. W., Zabinsky, Z. B., Serve, C. A. *Incorporating Weather Uncertainty in Airport Arrival Rate Decisions.* FAA-NEXTOR-INFORMS Conference on Air Traffic Management and Control. 2003.

Appendix A – Model Source Code

A.1. FCFS

Algorithm used:

Create SA(i) ! scheduled arrival for flight i

Where SA(i) = SA(j), assign SAC(i) = SA(i) + U[0,1]
Else SAC(i) = SA(i) ! U[0,1] is a uniform random variable

Sort flight data based on SAC(i) and assign additional index j = 0 ..N based on sorted order

Set x(i:j=0) = SA(i:j=0) ! x(i) is the assigned landing time for flight i

For j=1 to N
x(i:j) = max { SA(i:j) , x(i:j-1) + sep(size(i:j-1), size(i:j)) }

! sep(a,b) is the separation matrix based on plane size
end

A.2. P_delay

```
TITLE
    paxdelay;     ! min pax delay

DATA
      numB := EXCELRANGE("batchSize.xls", "batch");   ! sched arrival

INDEX
      i  := 0..numB;    !  planes
      j  := i;

      dataFields := 1..5;

      size  := (H, L, M, S);
      size1  := size;
      size2  := size;
DATA
      dataR[i,dataFields] := DATAFILE("lga.csv");
      SA[i] := dataR[i,2];
      S[i]  := dataR[i,5];
      NP[i] := dataR[i,4];

      sep[size1,size2] :=  DATAFILE("sep.csv"); !separation matrix
based on flight data, 10 planes, 11 x 11

VARIABLE
      x[i]
            EXPORT TO EXCELRANGE("mplOutput.xls","slot");
      delay[i]
```

```
            EXPORT TO EXCELRANGE("mplOutput.xls","delay");
      lastplane;

BINARY VARIABLES
      y[i,j];

MACRO
      passdelay = sum(i: delay * NP);
MODEL
    MIN z = passdelay;

SUBJECT TO
      exclude[i,j>i]  : x[i] - x[j] >= sep[ size1:=S[j], size2:=S[i] ]
- 100000 y[i,j];
      exclude2[i,j>i]  : x[j] - x[i] >= sep[ size1:=S[i], size2:=S[j] ]
- 100000 (1 - y[i,j] );
      find_arrivaldelay[i]:   delay >= x - SA;
      x[0]=SA[0];
      pp[i] : x[i]>=SA[0];

      last_in[i>0]:     lastplane >= x;

BOUNDS
  latest[i]: delay <= 1800;
  earliest[i]:    x >= SA ;
END
```

A.3. V_delay

```
TITLE
    V_delay;      ! min vehicle delay

DATA
      numB := EXCELRANGE("batchSize.xls", "batch");   ! sched arrival

INDEX
      i  := 0..numB;    !  planes
      j  := i;

      dataFields := 1..5;

      size  := (H, L, M, S);
      size1  := size;
      size2  := size;
DATA
      dataR[i,dataFields] := DATAFILE("lga.csv");
      SA[i] := dataR[i,2];
      S[i]  := dataR[i,5];
      NP[i] := dataR[i,4];

      sep[size1,size2] :=  DATAFILE("sep.csv"); !separation matrix
based on flight data, 10 planes, 11 x 11

VARIABLE
      x[i]
            EXPORT TO EXCELRANGE("mplOutput.xls","slot");
      delay[i]
```

```
            EXPORT TO EXCELRANGE("mplOutput.xls","delay");
      lastplane;

BINARY VARIABLES
      y[i,j];

MACRO
  totalDelay = sum(i:delay);
MODEL
    MIN z = totaldelay;

SUBJECT TO
      exclude[i,j>i]  : x[i] - x[j] >= sep[ size1:=S[j], size2:=S[i] ]
- 100000 y[i,j];
      exclude2[i,j>i]  : x[j] - x[i] >= sep[ size1:=S[i], size2:=S[j] ]
- 100000 (1 - y[i,j] );
      find_arrivaldelay[i]:   delay >= x - SA;
      x[0]=SA[0];
      pp[i] : x[i]>=SA[0];

      last_in[i>0]:     lastplane >= x;

BOUNDS
  latest[i]: delay <= 1800;
  earliest[i]:    x >= SA ;
END
```

A.4. V_thrpt

```
TITLE
    V_thrpt;      ! max vehicle throughput

DATA
      numB := EXCELRANGE("batchSize.xls", "batch");   ! sched arrival

INDEX
      i  := 0..numB;    !  planes
      j  := i;

      dataFields := 1..5;

      size  := (H, L, M, S);
      size1  := size;
      size2  := size;
DATA
      dataR[i,dataFields] := DATAFILE("lga.csv");
      SA[i] := dataR[i,2];
      S[i]  := dataR[i,5];
      NP[i] := dataR[i,4];

      sep[size1,size2] :=  DATAFILE("sep.csv"); !separation matrix
based on flight data, 10 planes, 11 x 11

VARIABLE
      x[i]
            EXPORT TO EXCELRANGE("mplOutput.xls","slot");
      delay[i]
```

```
            EXPORT TO EXCELRANGE("mplOutput.xls","delay");
      lastplane;

BINARY VARIABLES
      y[i,j];

MACRO
  totalDelay = sum(i:delay);
MODEL
    MIN z = lastplane;

SUBJECT TO
      exclude[i,j>i]  : x[i] - x[j] >= sep[ size1:=S[j], size2:=S[i] ]
- 100000 y[i,j];
      exclude2[i,j>i]  : x[j] - x[i] >= sep[ size1:=S[i], size2:=S[j] ]
- 100000 (1 - y[i,j] );
      find_arrivaldelay[i]:   delay >= x - SA;
      x[0]=SA[0];
      pp[i] : x[i]>=SA[0];

      last_in[i>0]:     lastplane >= x;

BOUNDS
  latest[i]: delay <= 1800;
  earliest[i]:    x >= SA ;
END
```

A.5. A_fair

```
TITLE

    FA;     ! fcfs rule with airline fairness opt, data preprocessed

INDEX

      i  := EXCELRANGE("lga.xls", "index");     ! 0 is the start,
planes
      j  := i;
      a  := EXCELRANGE("lga_catable","airlines");

      size  := (H, B7, L, S);
      size1  := size;
      size2  := size;

      ca_fl[a,i] :=  (COM,3 DAL,4 USA,5 CHQ,9 CJC,10 EGF,11 CHQ,14
CHQ,15 CJC,16 COM,17 CJC,21 CJC,22 EGF,23 CJC,29 PDT,30 ACA,32 EGF,33
EGF,34 COM,35 EGF,36 TRS,37 CHQ,40 USA,41 NWA,49 COM,54 DAL,55 EGF,56
CJC,61 USA,62 AAL,64 JBU,65 COM,70 USA,71 EGF,73 UAL,74 AMT,80 COM,86
DAL,87 CHQ,88 CHQ,89 CJC,90 COM,91 EGF,92 AAL,97 AAL,98 DAL,99 NWA,101
PDT,102 AAL,108 COA,109 DAL,110 NKS,111 PDT,112 ACA,115 EGF,116 USA,118
USA,119 CHQ,126 TRS,127 AAL,129 DAL,130 EGF,131 EGF,132 ACA,136 CJC,137
EGF,138 CJC,141 MEP,142 UAL,146 USA,147 AAL,152 AAL,153 CJC,157 CJC,158
DAL,159 DAL,160 DAL,161 EGF,162 AAL,167 CJC,171 NWA,172 CHQ,176 CHQ,177
AAL,179 CJC,180 CJC,181 COM,182 DAL,183 PDT,184 AAL,186 USA,187 CHQ,195
CHQ,196 CJC,198 COM,199 DAL,200 PDT,201 ACA,205 CJC,206 JBU,207 EGF,214
AAL,218 AMT,219 COM,220 DAL,221 EGF,222 NWA,223 UAL,226 USA,227 CHQ,234
CHQ,235 COA,236 DAL,237 DAL,238 DAL,239 AAL,240 AAL,241 CJC,242 PDT,243
```

```
AAL,245 ACA,246 COM,248 PDT,249 PDT,253 CJC,254 COM,255 DAL,256 DAL,257
EGF,258 EGF,259 NWA,260 CJC,262 UAL,263 AAL,272 DAL,273 AAL,276 COM,277
CJC,279 COM,280 DAL,281 DAL,282 TRS,283 EGF,284 EGF,285 EGF,286 PDT,287
PDT,289 UAL,290 CJC,297 CJC,298 NKS,299 CHQ,300 CHQ,301 CJC,302 COM,303
EGF,304 ACA,306 USA,307 DAL,310 TRS,311 AAL,319 CJC,320 EGF,321 EGF,322
EGF,323 NKS,324 DAL,326 JBU,327 AAL,328 EGF,329 CHQ,334 CJC,335 DAL,336
USA,337 AAL,338 AAL,339 EGF,340 EGF,341 UAL,342 PDT,344 UAL,345 CHQ,348
USA,349 AAL,352 DAL,353 MEP,354 ACA,358 AAL,363 ACA,364 COM,365 MEP,366
CHQ,369 CJC,370 COM,371 DAL,372 PDT,373 DAL,374 EGF,375 AAL,376 USA,377
EGF,380 AAL,383 AAL,384 CHQ,387 EGF,388 UAL,389 UAL,390 CJC,392 DAL,393
AAL,399 USA,400 CJC,402 CJC,403 COA,404 DAL,405 DAL,406 EGF,407 NKS,408
COM,412 PDT,413 AAL,421 AAL,422 CHQ,423 COM,427 DAL,428 EGF,430 NWA,431
COA,437 USA,438 AAL,448 CHQ,449 MEP,450 AAL,465 DAL,466 JBU,467 EGF,469
NWA,470 PDT,471 TRS,472 DAL,473 USA,474 AAL,479 JBU,480 AAL,482 NKS,483
USA,484 DAL,489 DAL,490 CJC,494 MEP,495 AAL,496 NWA,497 AAL,500 COA,501
NKS,505 NWA,506 AAL,511 AAL,519 AMT,520 UAL,521);

DATA
      SA[i] := EXCELRANGE("lga.xls", "arrival");      ! sched arrival

      S[i] := EXCELRANGE("lga.xls", "size");    ! size from 1=heavy

      NP[i] := EXCELRANGE("lga.xls", "pax");    ! no. passengers


      NPC[a] := EXCELRANGE("lga_catable.xls", "paxdata");   ! total
passengers for airline


      sep[size1,size2]  :=  DATAFILE("sep.csv");      !separation
matrix based on flight data, 10 planes, 11 x 11


VARIABLE

      x[i]
                  EXPORT TO EXCELRANGE("lga.xls", "slot");

                  ! landing slot time, as of start of final approach

      delay[i]
                  EXPORT TO EXCELRANGE("lga.xls", "delay");

                  !arrival delay from schedule for flight

      capen[a]
                  EXPORT TO EXCELRANGE("lga_catable.xls", "delay");

                  !pax delay fraction for carrier a

      maxpen;

      lastplane;

BINARY VARIABLES

      y[i,j];
```

```
MACRO
      passdelay = sum(i: delay * NP);
      totalpen = sum(capen);

MODEL
    MIN maxpen + totalpen;

SUBJECT TO
      exclude[i,j>i] WHERE SA[i] < SA[j] : x[j] - x[i] >= sep[
size1:=S[i], size2:=S[j] ];

      exclude2[i,j>i] WHERE SA[i] = SA[j] : x[j] - x[i] >= sep[
size1:=S[i], size2:=S[j] ] - 10000 y[i,j];

      exclude3[i,j>i] WHERE SA[i] = SA[j] : x[i] - x[j] >= sep[
size1:=S[j], size2:=S[i] ] - 10000 (1 - y[i,j] );

      find_arrivaldelay[i]:   delay = x - SA;

      last_in[i]:       lastplane >= x;

      capenalty[a]:     capen = 1/NPC * sum(i IN ca_fl : delay * NP) ;

      maxpenalty[a]:    maxpen >= capen ;

BOUNDS
      earliest[i]:      x >= SA ;  !

      0 <= delay  <= 1800;

END
```

A.6. WCG

```
/*
 * Flight.java
 *
 * Created on April 23, 2007, 3:29 PM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

package databasetest1;

/**
 *
 * @author Jeffrey
 */
public class Flight {

    private int flightIndex;
    private String flightNumber;
    private int scheduledArrivalTime;
```

```
    private String aircraftType;
    private String carrier;
    private int numPassengers;
    private WeightClass weightClass;
    private int actualArrivalTime;
    private int delay;

    /** Creates a new instance of Flight */
    public Flight() {
    }

    public Flight(int flightIndex, int scheduledArrivalTime, String
carrier, int numPassengers, WeightClass weightClass) {
        this.setFlightIndex(flightIndex);
        this.setScheduledArrivalTime(scheduledArrivalTime);
        this.setCarrier(carrier);
        this.setNumPassengers(numPassengers);
        this.setWeightClass(weightClass);
    }

    public int getFlightIndex() {
        return flightIndex;
    }

    public void setFlightIndex(int flightIndex) {
        this.flightIndex = flightIndex;
    }

    public String getFlightNumber() {
        return flightNumber;
    }

    public void setFlightNumber(String flightNumber) {
        this.flightNumber = flightNumber;
    }

    public int getScheduledArrivalTime() {
        return scheduledArrivalTime;
    }

    public void setScheduledArrivalTime(int scheduledArrivalTime) {
        this.scheduledArrivalTime = scheduledArrivalTime;
    }

    public String getAircraftType() {
        return aircraftType;
    }

    public void setAircraftType(String aircraftType) {
        this.aircraftType = aircraftType;
    }

    public String getCarrier() {
        return carrier;
    }

    public void setCarrier(String carrier) {
```

```
        this.carrier = carrier;
    }

    public int getNumPassengers() {
        return numPassengers;
    }

    public void setNumPassengers(int numPassengers) {
        this.numPassengers = numPassengers;
    }

    public WeightClass getWeightClass() {
        return weightClass;
    }

    public void setWeightClass(WeightClass weightClass) {
        this.weightClass = weightClass;
    }

    public int getActualArrivalTime() {
        return actualArrivalTime;
    }

    public void setActualArrivalTime(int actualArrivalTime) {
        this.actualArrivalTime = actualArrivalTime;
    }

    public int getDelay() {
        return delay;
    }

    public void setDelay(int delay) {
        this.delay = delay;
    }

    public void updateDelay() {
        setDelay(getActualArrivalTime()-getScheduledArrivalTime());
    }

    public String toString() {
        return (scheduledArrivalTime + ", " + weightClass.toString() +
", " + actualArrivalTime + ", " + delay);
    }
}
/*
 * WeightClass.java
 *
 * Created on April 23, 2007, 3:38 PM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

package databasetest1;

import java.io.BufferedReader;
import java.io.File;
```

```
import java.io.FileReader;
import java.io.IOException;
import java.util.Scanner;

/**
 *
 * @author Jeffrey
 */
public enum WeightClass {
    SMALL,
    LARGE,
    B757,
    HEAVY;

    private int beforeSmall;
    private int beforeLarge;
    private int beforeB757;
    private int beforeHeavy;

    public static final void initialization(File dataFile) throws
IOException {
        BufferedReader inputStream = null;
        String line;
        Scanner s = null;
        try {
            inputStream =
                new BufferedReader(new FileReader(dataFile));
            for (WeightClass item : WeightClass.values()) {
                if ((line = inputStream.readLine()) != null) {
                    s = new Scanner(line);
                }
                item.beforeSmall = s.nextInt();
                item.beforeLarge = s.nextInt();
                item.beforeB757 = s.nextInt();
                item.beforeHeavy = s.nextInt();
            }

        } finally {
            if (inputStream != null) {
                inputStream.close();
            }
        }
        s.close();
    }

    public static WeightClass getWeightClassFromString (String
weightClassString) {
        WeightClass weightClass = LARGE;
        if (weightClassString.equals(WeightClass.SMALL.toString())) {
            weightClass = SMALL;
        } else if
(weightClassString.equals(WeightClass.LARGE.toString())) {
            weightClass = LARGE;
        } else if
(weightClassString.equals(WeightClass.B757.toString())) {
            weightClass = B757;
```

```
        } else if
(weightClassString.equals(WeightClass.HEAVY.toString())) {
            weightClass = HEAVY;
        } else System.err.println("wrong Wake Class String.");
        return weightClass;
    }

    public static WeightClass getWeightClassFromInt (int
weightClassInt) {
        WeightClass weightClass = LARGE;
        if (weightClassInt == 4) {
            weightClass = SMALL;
        } else if (weightClassInt == 3) {
            weightClass = LARGE;
        } else if (weightClassInt == 2) {
            weightClass = B757;
        } else if (weightClassInt == 1) {
            weightClass = HEAVY;
        } else System.err.println("wrong Wake Class Int.");
        return weightClass;
    }

    int getSeparation (WeightClass weightClass) {
        int separation=0;
        switch (weightClass) {
            case SMALL: separation = beforeSmall; break;
            case LARGE: separation =  beforeLarge; break;
            case B757: separation = beforeB757; break;
            case HEAVY: separation = beforeHeavy; break;
            default: System.out.println("Invalid weightClass."); break;
        }
        return separation;
    }

    int convertedOrdinal (WeightClass desiredWeightClass, Boolean
wantLarger) {
        int relativeOrdinal = this.ordinal() -
desiredWeightClass.ordinal();
        int convertedOrdinal = Math.abs(relativeOrdinal);
        if ((wantLarger && relativeOrdinal<0) || (!wantLarger &&
relativeOrdinal>0)) {
            convertedOrdinal += 3;
        }
        return convertedOrdinal;
    }

    boolean isBetterThan(WeightClass baseWeightClass, WeightClass
desiredWeightClass, Boolean wantLarger) {
        if (this.convertedOrdinal(desiredWeightClass, wantLarger) <
baseWeightClass.convertedOrdinal(desiredWeightClass, wantLarger)) {
            return true;
        } else {
            return false;
        }
    }
}
/*
```

```
 * Main.java
 *
 */

package databasetest1;

import java.io.File;
import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.LinkedList;
import java.util.ListIterator;

/*
 * Main.java
 *
 * Created on April 4, 2007, 8:36 PM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

package databasetest1;

import java.io.File;
import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.LinkedList;
import java.util.ListIterator;

/**
 *
 * @author Jeffrey
 */
public class Main {

    /** Creates a new instance of Main */
    public Main() {
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) throws IOException {
        WeightClass.initialization(new File("sep from s to h
int.txt"));
        LinkedList<Flight> flights = new LinkedList();
        try {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

```
            Connection c =
DriverManager.getConnection("jdbc:odbc:or680excel", "","");
            Statement stmnt = c.createStatement();
            String query = "select * from [Sheet1$]";
            ResultSet rs = stmnt.executeQuery(query);
            while (rs.next()) {
                flights.add(new Flight(rs.getInt("index"),
rs.getInt("sched"), rs.getString("carrier"),
rs.getInt("pax"),WeightClass.getWeightClassFromInt(rs.getInt("size"))
));
            }
            stmnt.close();
            c.close();
        } catch (SQLException s) {
            System.out.println("SQL Error: " + s.toString() + " " +
s.getErrorCode() + " " + s.getSQLState());
            System.exit(0);
        } catch (Exception e) {
            System.out.println("Error: " + e.toString() +
e.getMessage());
            System.exit(0);
        }
        ListIterator<Flight> flightIterator = flights.listIterator();
        LinkedList<Flight> flightsSequenced = new LinkedList();
        Flight previousFlight = null;
        Flight iteratedFlight = null;
        if (flightIterator.hasNext()) {
            iteratedFlight = flightIterator.next();

iteratedFlight.setActualArrivalTime(iteratedFlight.getScheduledArrivalT
ime());
            iteratedFlight.updateDelay();
            flightIterator.remove();
            flightsSequenced.add(iteratedFlight);
            previousFlight = iteratedFlight;
        } else {
            System.out.println("Empty schedule!");
            return;
        }
        Boolean wantLarger = true;
        while (flightIterator.hasNext()) {
            int earliestArrivalTimeForSmall =
previousFlight.getActualArrivalTime() +
previousFlight.getWeightClass().getSeparation(WeightClass.SMALL);
            Flight bestEligibleFlight = null;
            do {
                iteratedFlight = flightIterator.next();
                if (iteratedFlight.getScheduledArrivalTime()<=
previousFlight.getActualArrivalTime() +
previousFlight.getWeightClass().getSeparation(iteratedFlight.getWeightC
lass())) {
                    if (bestEligibleFlight == null ||
iteratedFlight.getWeightClass().isBetterThan(bestEligibleFlight.getWeig
htClass(), previousFlight.getWeightClass(), wantLarger)) {
                        bestEligibleFlight = iteratedFlight;
                    }
                }
```

```
            } while (iteratedFlight.getScheduledArrivalTime() <=
earliestArrivalTimeForSmall && flightIterator.hasNext());
            if (bestEligibleFlight == null) {
                wantLarger = true;
                flightIterator = flights.listIterator();
                iteratedFlight = flightIterator.next();

iteratedFlight.setActualArrivalTime(iteratedFlight.getScheduledArrivalT
ime());
            } else {
                if
(bestEligibleFlight.getWeightClass().convertedOrdinal(previousFlight.ge
tWeightClass(), wantLarger) > 3) {
                    wantLarger = !wantLarger;
                }
                flightIterator =
flights.listIterator(flights.indexOf(bestEligibleFlight));
                iteratedFlight = flightIterator.next();

iteratedFlight.setActualArrivalTime(previousFlight.getActualArrivalTime
() +
previousFlight.getWeightClass().getSeparation(iteratedFlight.getWeightC
lass()));
            }
            iteratedFlight.updateDelay();
            flightIterator.remove();
            flightsSequenced.add(iteratedFlight);
            previousFlight = iteratedFlight;
            flightIterator = flights.listIterator();
        }
        flightIterator = flightsSequenced.listIterator();
        int totalNumFlights = 0;
        int totalFlightDelay = 0;
        int totalNumPassengers = 0;
        int totalPassengerDelay = 0;
        while (flightIterator.hasNext()) {
            iteratedFlight = flightIterator.next();
            totalNumFlights++ ;
            totalFlightDelay += iteratedFlight.getDelay();
            totalNumPassengers += iteratedFlight.getNumPassengers();
            totalPassengerDelay += iteratedFlight.getDelay() *
iteratedFlight.getNumPassengers();
        }
        double averageFlightDelay = (double)(totalFlightDelay) /
totalNumFlights;
        double averagePassengerDelay = (double) (totalPassengerDelay) /
totalNumPassengers;
        System.out.println("Number of flights: " + totalNumFlights);
        System.out.println("Average flight delay in minutes: " +
averageFlightDelay/60);
        System.out.println("Number of passengers: " +
totalNumPassengers);
        System.out.println("Average passenger delay in minutes: " +
averagePassengerDelay/60);
        try {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

```
            Connection c =
DriverManager.getConnection("jdbc:odbc:or680excel", "","");
            Statement stmnt = c.createStatement();
            stmnt.executeUpdate("DROP TABLE [output]");
            stmnt.executeUpdate("CREATE TABLE [output] (flightIndex
INTEGER, " +
                    "scheduledArrivalTime INTEGER, numPassengers
INTEGER, " +
                    "weightClass VARCHAR, actualArrivalTime INTEGER,
delay INTEGER, newSequence INTEGER)");
            int newSequence = 0;
            for (Flight item: flightsSequenced) {
                int thisFlightIndex = item.getFlightIndex();
                String thisFlightNumber = item.getFlightNumber();
                String query = "INSERT INTO [output] (flightIndex,
scheduledArrivalTime, numPassengers, " +
                        "weightClass, actualArrivalTime, delay,
newSequence) VALUES ("
                        + item.getFlightIndex() +
                        "," + item.getScheduledArrivalTime() +
                        "," + item.getNumPassengers() +
                        ",'" + item.getWeightClass() + "'" +
                        "," + item.getActualArrivalTime() +
                        "," + item.getDelay() +
                        "," + newSequence++ +
                        ")";
                stmnt.executeUpdate(query);
            }
            String query = "select * from [Sheet1$]";
            ResultSet rs = stmnt.executeQuery(query);
            stmnt.close();
            c.close();
        } catch (SQLException s) {
            System.out.println("SQL Error: " + s.toString() + " " +
s.getErrorCode() + " " + s.getSQLState());
            System.exit(0);
        } catch (Exception e) {
            System.out.println("Error: " + e.toString() +
e.getMessage());
            System.exit(0);
        }
    }

}
```

A.7. Windowing Code

myExcelSep.cpp

(This is the main code which calls the optimization codes and also does the excel sheet manipulations)

```
#include "conio.h"
#include "BasicExcelVC6.cpp"
#include <time.h>

using namespace YExcel;
```

```
double* mainOfOptimax(int,char*);

int max(int a, int b) {
    int maxval;
    if (a >= b) {
        maxval = a;
    } else {
        maxval = b;
    }
    return maxval;
}//end max

int main(int argc, char* argv[])
{
  int totalFlights = 523;
  char *modelName = ".\\T.mpl";
  size_t one = 0;  // variable type to open first worksheet of any
excel file
  int inputDataColumns = 5; // number of fields in the FlightData input
file
                                          // index, arrival time,, exp
arr time, size, PAX

  int batch;
  time_t start,end;
  double dif;

  BasicExcel eIndex;
  eIndex.Load("batchSize.xls");
  BasicExcelWorksheet* sheet_index = eIndex.GetWorksheet(one);
  batch = sheet_index->Cell(0,0)->GetInteger();

  // Load the workbook FLIGHTDATA with one sheet 'data', this contains
all the flight data
  BasicExcel e;
  e.Load("flightData.xls");
  BasicExcelWorksheet* sheet_fData = e.GetWorksheet(one);


  //create another worksheet, FinalOutput which contains the final seq
of flights
  BasicExcel e3;
  e3.New(1);
  BasicExcelWorksheet* sheet_finalOutput = e3.GetWorksheet(one);

  //first row of the input file FLIGHTDATA is the header, second row is
the leading flight(already fixed)
 // Load the workbook lga which contains the input for this iteration
  BasicExcel e1;
  e1.New(1);
  BasicExcelWorksheet* sheet_lga = e1.GetWorksheet(one);

      for (size_t r=0;r<batch+1;++r)
      {
            for (size_t c=0;c<inputDataColumns;++c)
                  {
```

```
                        //sheet_lga->Cell(r+1,c)-
>SetDouble(sheet_fData->Cell(r+1,c)->GetDouble());
                        sheet_lga->Cell(r,c)->SetDouble(sheet_fData-
>Cell(r+1,c)->GetDouble());
                        if(r==0){
                                                sheet_finalOutput->Cell(1,c)-
>SetDouble(sheet_fData->Cell(r+1,c)->GetDouble());
                                               }
                  }
            if(r==0)
                  {
                        sheet_finalOutput->Cell(0,0)-
>SetString("Index");
                        sheet_finalOutput->Cell(0,1)-
>SetString("Arrival");
                        sheet_finalOutput->Cell(0,2)->SetString("E
Arrival");
                        sheet_finalOutput->Cell(0,3)-
>SetString("Type");
                        sheet_finalOutput->Cell(0,4)->SetString("PAX");
                        sheet_finalOutput->Cell(0,5)->SetString("Sched
Arr");
                        sheet_finalOutput->Cell(0,6)-
>SetString("Delay");
                  }
      }

        size_t maxRows = sheet_lga->GetTotalRows();
    size_t maxCols = sheet_lga->GetTotalCols();

        e1.SaveAs("lga.xls");
        e3.SaveAs("finalOutput.xls");

        ofstream f("lga.csv");
    sheet_lga->Print(f, ',', '\"'); // Save the lga.xls file as lgs.CSV
    f.close();

        size_t rr=0;
        size_t cc=0;
        BasicExcel eOut;

        int num;
        int leadIndex;
          double leadSlot;
          double leadDelay;
        num= max(totalFlights-batch,0);
        //num=1;
        time (&start);
        for (int iter = 0 ; iter < num ; ++iter)
        {
          // general comments: Call the MPL program.. it will return an
integer which is the index
          // of the Flight which will be first in the given sequence
(after the leading flight. index=0
          // lets say that index is leadIndex

          double *ret;
```

```
        ret = mainOfOptimax(batch,modelName);

        leadIndex = ret[0];
        leadSlot  = ret[1];
        leadDelay = ret[2];

        if(iter!=num-1)
        {
              for (cc=0;cc<inputDataColumns;++cc) //because 2 new
columns(sched arr, delay)
                  {
                        sheet_lga->Cell(0,cc)->SetDouble(sheet_lga-
>Cell(leadIndex,cc)->GetDouble());
                        sheet_finalOutput->Cell(iter+2,cc)-
>SetDouble(sheet_lga->Cell(leadIndex,cc)->GetDouble());
                        // replace the winning flight row with the next
row of our database(flightData)
                        sheet_lga->Cell(leadIndex,cc)-
>SetDouble(sheet_fData->Cell(iter+batch+2,cc)->GetDouble());
                  }

            // sarrival data: column 1
      sheet_lga->Cell(0,1)->SetDouble(leadSlot);
      //because 2 new columns(sched arr, delay)
      sheet_finalOutput->Cell(iter+2,inputDataColumns)-
>SetDouble(leadSlot);
      sheet_finalOutput->Cell(iter+2,inputDataColumns+1)-
>SetDouble(leadDelay);
        }
        else
        {
                time (&end);
                  dif = difftime (end,start);

                      BasicExcel ee;
                  ee.Load("mplOutput.xls");
                  BasicExcelWorksheet* sheet_mplOutput =
ee.GetWorksheet(one);

                  sheet_finalOutput->Cell(1,8)->SetDouble(dif);


                  for (int ii=0;ii<batch;++ii)
                  {
                        for (cc=0;cc<inputDataColumns+2;++cc) //because
2 new columns(sched arr, delay)
                        {
                              if(cc<inputDataColumns)
                              {
                                    sheet_finalOutput-
>Cell(num+1+ii,cc)->SetDouble(sheet_lga->Cell(ii+1,cc)->GetDouble());

                              }
                              else
                              {
```

```
                                        sheet_finalOutput-
>Cell(num+1+ii,cc)->SetDouble(sheet_mplOutput->Cell(ii+1,cc-
inputDataColumns)->GetDouble());
                                }
                        }
                }
        }

      e1.SaveAs("lga.xls");
      ofstream f("lga.csv");
      sheet_lga->Print(f, ',', '\"'); // Save the lga.xls file as
lgs.CSV
      f.close();

      e3.SaveAs("finalOutput.xls");
      //getch();
      }

printf ("It took %.2lf seconds to RUN \n", dif );
      return 0;
}
```

myMPL.cpp

(This is the code which is called by the above script(myExcelSep.cpp). This does the actual calling of the MPL optimization code.)

```
 /*  OMaxTest.cpp  */

#include <stdio.h>
#include <iostream.h>
#include <tchar.h>
#include<conio.h>

#include <windows.h>
#include <atlbase.h>

#import "OptiMax.tlb"

int minIndex;
int batchS;
double *combined;

int minFunction(double *a,int size)
{
        int i;
        int index;
        double temp=99999;
        for (i=1;i<=size;i++)
        {
                if(a[i]<temp)
```

```
                {
                        temp=a[i];
                        index = i;
                }
        }
        cout<<a[index];
        return index;
}

void SolveModelTypeLib(char *ModelFilename, char *SolverName)
{

        using namespace MPLLib;

        printf("Call OptiMax using TypeLib\n");
        try {
                IOptiMaxPtr     pMpl( __uuidof(MPLLib::OptiMax));

                ISolverPtr                              pSolver;
                IModelPtr                               pModel;
                IMatrixPtr                              pMatrix;
                IVariablesPtr                   pVars;
                IConstraintsPtr         pCons;
                ISolutionPtr                    pSol;
                IVariableVectorPtr      pVarVector;
                IVariableVectorPtr      pVarVector1;
                IVariablePtr                    pVar;
                IVariablePtr                    pVar1;

                long      result;

                pSolver = pMpl->Solvers->Add(SolverName);      // Set pSolver =
MPL.Solvers.Add "CPLEX"

                pModel = pMpl->Models->Add("Model1");                  // Set pModel =
MPL.Models.Add("Model1")

           printf("READ:  '%s'\n", ModelFilename);
                result = pModel->ReadModel(ModelFilename);             // Set result =
pModel.ReadModel("planning.mpl")

                if (result) {
                        printf("ReadModel(%s) failed (result=%d\n\n", ModelFilename, result);
                        return;
                }

                pMatrix = pModel->Matrix;
        // Set pMatrix = pModel.Matrix
                pVars = pMatrix->Variables;
        // Set pVars = pMatrix.Variables
```

```
		pCons = pMatrix->Constraints;
	// Set pCons = pMatrix.Constraints

	//	printf("MODEL: vars=%d, cons=%d, nz=%d, int=%d\n",
	//	                                        pVars->Count, pCons->Count, pMatrix-
>NonZeroCount, pVars->IntegerCount);

		result = pModel->Solve(pSolver);
	// Set result = pModel.Solve(pSolver)
		if (result) {
			printf("Solve() failed (result=%d\n\n", result);
			return;
		}

		pSol = pModel->Solution;
	// Set pSol = pModel.Solution

		printf("SOLVE: obj=%.10lg, iter=%d, nodes=%d, result='%s'\n",
							pSol->ObjectValue, pSol->IterationCount,
							pSol->NodeCount, (LPCSTR)pSol-
>ResultString);

		pVarVector   = pModel->VariableVectors->GetItem("delay");     // pVarVector =
pModel.VariableVectors("Production")
		pVarVector1  = pModel->VariableVectors->GetItem("x");         // pVarVector =
pModel.VariableVectors("Production")

		int i = 0;

		/* read this from excel file */
		/* int total = 8; (# of flights = total+1 )*/
// batch is a global variable = batch size

		//batchS = 10;

		double *delays;
		double *xx;
		delays = (double *) malloc((batchS+1)*sizeof(double));
		memset(delays,0,(batchS+1)*sizeof(double));
	//	//double delays[11];

		xx = (double *) malloc((batchS+1)*sizeof(double));
		memset(xx,0,(batchS+1)*sizeof(double));

		combined = (double *) malloc(3*sizeof(double));
		memset(combined,0,3);


		printf("DELAY: ");
		pVar = pVarVector->MoveFirstPos();
		while (pVarVector->PosValid) {
```

```
                delays[i]=pVar->Activity;
                i++;
                printf(" [%d]=%.10lg,", i, pVar->Activity);        // pVar.Activity
                cout<<"\n"<<i<<"\n";
                pVar = pVarVector->MoveNextPos();
        }
        printf("\n\n");

        i=0;
        printf("X(Slots): ");
        pVar1 = pVarVector1->MoveFirstPos();
        while (pVarVector1->PosValid) {
                xx[i]=pVar1->Activity;
                i++;
                printf(" [%d]=%.10lg,", i, pVar1->Activity);       // pVar.Activity
                cout<<"\n"<<i<<"\n";
                pVar1 = pVarVector1->MoveNextPos();
        }
        printf("\n\n");



        /* exclude the first element */
        // 0 is the header row
        for (i=0; i<=batchS; i++)
        {                       cout<<i<<"\t"<<xx[i]<<"\n";             }

        for (i=0; i<=batchS; i++)
        {                       cout<<i<<"\t"<<delays[i]<<"\n";                 }



        minIndex=minFunction(xx,batchS);
        combined[0] = minIndex;
        combined[1] = xx[minIndex];
        combined[2] = delays[minIndex];

    }
    catch (const _com_error& Err) {
        printf("Error: %s  (0x%x)\n\n%s\n",
                (LPCSTR)Err.ErrorMessage(), Err.Error(),
(LPCSTR)Err.Description());
    }
}

double* mainOfOptimax(int x, char* modN)
{
    batchS = x;
    char *SolverName = "d:\\mplwin4\\cplex91.dll";
    CoInitialize(NULL);
    int mm=0;
```

```
	cout<<"\n"<<"popo"<<"\n";
	cout<<minIndex<<"\n";
	SolveModelTypeLib(modN, SolverName);

	cout<<minIndex<<"\n";
	cout<<combined[0]<<"\n";
	cout<<combined[1]<<"\n";
	cout<<combined[2]<<"\n";
	cout<<"\n"<<"popo"<<"\n";
	CoUninitialize();
	//return minIndex;
	return combined;
	//return 0;
}


/*
int main()
{
	batchS = 10;

	char *SolverName = "d:\\mplwin4\\cplex91.dll";
	CoInitialize(NULL);
	int mm=0;
	cout<<"\n"<<"popo"<<"\n";
	cout<<minIndex<<"\n";
	//SolveModelTypeLib("C:\\Documents and
Settings\\Administrator\\Desktop\\680_update\\opti\\FT.mpl", SolverName);
	SolveModelTypeLib(".\\FT.mpl", SolverName);
	cout<<minIndex<<"\n";
	cout<<"\n"<<"popo"<<"\n";
	CoUninitialize();
	return minIndex;
	//return 0;
}
*/
```

Appendix B – Work Breakdown Structure

B.1 – Vivek Kumar

- Prepared project proposal presentation and draft
- Researched already existing work in the field
- Explored Optimax library for J#
- Implemented windowing in C++
- Used BasicExcel(open source) library for excel manipulations through C++
- Assisted with paper and presentation

B.2 – David Teale

- Solved FCFS case using Excel
- Formulated MPL models
- Ran and checked mpl models
- Assisted with integration of windowing and mpl models
- Massaged results
- Analyzed results
- Assisted with presentation and paper

B.3 – Jianfeng Wang

- Assisted with problem definition and formulation
- Collected input data
- Solved FCFS using Java
- Used Java Database Connectivity (JDBC) to access Excel file
- Implemented Weight Class Grouping using Java
- Assisted with presentation and paper

B.4 – Seth Wenchel

- Organized tasking of group members
- Researched previous work
- Worked with Optimax and J#
- Created windowing approach
- Ran MPL models
- Collected data
- Assisted with presentation and paper