A STUDY OF HEURISTIC APPROACHES FOR RUNWAY SCHEDULING FOR THE DALLAS-FORT WORTH AIRPORT

A Thesis

by

PAUL WILLIAM STIVERSON

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

May 2009

Major Subject: Mechanical Engineering

A STUDY OF HEURISTIC APPROACHES FOR RUNWAY SCHEDULING FOR THE DALLAS-FORT WORTH AIRPORT

A Thesis

by

PAUL WILLIAM STIVERSON

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

Chair of Committee,	Sivakumar Rathinam
Committee Members,	Darbha Swaroop
	Kumbakonam Rajagopal
	Sergiy Butenko
Head of Department,	Dennis O'Neal

May 2009

Major Subject: Mechanical Engineering

ABSTRACT

A Study of Heuristic Approaches for Runway Scheduling
for the Dallas-Fort Worth Airport. (May 2009)
Paul William Stiverson, B.S., Texas A&M University
Chair of Advisory Committee: Dr. Sivakumar Rathinam

Recent work in air transit efficiency has increased en-route efficiency to a point that airport efficiency is the bottleneck. With the expected expansion of air transit it will become important to get the most out of airport capacity. Departure scheduling is an area where efficiency stands to be improved, but due to the complicated nature of the problem an optimal solution is not always forthcoming. A heuristic approach can be used to find a sub-optimal take-off order in a significantly faster time than the optimal solution can be found using known methods.

The aim of this research is to explore such heuristics and catalog their solution characteristics. A greedy approach as well as a k-interchange approach were developed to find improved takeoff sequences. When possible, the optimal solution was found to benchmark the performance of the heuristics, in general the heuristic solutions were within 10–15% of the optimal solution. The heuristic solutions showed improvements of up to 15% over the first-in first-out order with a running time around 4 ms.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

There has been significant interest in problems related with air traffic control as air transportation plays a critical role in the growth of the U.S. Economy. The Joint Economic Committee (JEC) of the United States Senate estimated that air transit delays caused 41 billion dollars of economic losses in 2007 alone.[1] The monetary losses stem from increases in operating costs on the part of the airlines, and wasted time on the part of the passenger. Apart from the monetary losses delay also causes an increase in fuel consumption and emissions as well as degrading the passenger experience. There are a number of factors that cause delay, chief among these is weather. Some factors, like mechanical troubles, cause a localized delay that doesn't necessarily affect other aircraft. It is reported that 20% of all delay occurs as aircraft taxi between the gates and the runway.[1] Currently, human controllers determine the schedule of the aircraft at the runways Real-time simulation tools could assist human controllers and improve the efficiency of the taxiways.

The research presented in this thesis focuses on improving the order in which the aircraft are allowed to utilize a departure runway subject to timing, separation, and ordering constraints. The takeoff orders generated using the algorithms explained within are tested against the objective functions described below.

Total Delay The sum of the delays incurred by all aircraft, this measures how many plane-hours are being spent in the queue. The delay is defined as the amount of time between the aircraft's earliest departure time and its proposed departure time.

The journal model is *IEEE Transactions on Automatic Control.*

Makespan The minimizing the maximum take-off time, this objective measures how long it takes push all the aircraft through the queueing area (into the air).

The timing constraint requires that the departure time of an aircraft must at least be equal to the earliest available time of the aircraft, that is to say: an aircraft cannot depart until it has reached the runway.

A. Separation Constraints

The efficiency of a takeoff sequence depends primarily on two factors: First, the aircraft type; and second the aircraft destination (or departure fix). There is a required separation between the departure times of any two aircraft based on their sizes and weights, this separation needs to be enforced for the sake of safety and to preserve the condition of the runway surface. Consider, for instance, scheduling a Boeing 747 and a short range turbo-prop aircraft. If the 747 were allowed to take off first then the smaller aircraft would be forced to wait for the vortices generated by the large aircraft to dissipate. However, if the order were reversed the separation between the aircraft would be reduced since the larger craft will not be significantly affected by the wake of the small aircraft.

The second factor to consider when generating a takeoff sequence deals with airspace restrictions. Often, due to weather or destination airport constraints, there is a need to limit the throughput of a specific airspace. For safety and fuel savings it is preferable to meter the aircraft while on the ground rather than in the air, so—if an aircraft is departing toward airspace that is subject to a throughput restriction—its departure time will be dependent on the previous aircraft that departed toward that airspace. The direction in which an aircraft heads immediately after takeoff is known as a "fix". In the research presented in this thesis these airspace restrictions

are referred to as “fix constraints”.

B. Runway Layout

This paper will focus on departure scheduling for a single runway at the DFW airport.

Each departure runway has associated with it a series of queues where aircraft are supposed to wait for takeoff clearance. It is assumed for this research that once an aircraft enters a queue it is not permitted to change queues, or otherwise leave the queue until it enters the runway for takeoff. Once on the runway it cannot be re-queued. The model used considers 3 queues.

The aircraft enters the queues from the general taxiway; the geometry of the taxiway is complicated and outside the scope of the paper, however it will be assumed that there are three taxiways that can feed each of the queues equally. For the remainder of the paper these taxiways will be referred to as “feeders”. Figure 1 shows the configuration of the pertinent parts of the DFW airport, note that many of the details have been omitted and the positions have been modified for clarity.

Due to the layout of the airport it is possible (and often necessary) for arriving aircraft to cross the departure runway. So, while focusing on departures it would be shortsighted to neglect arrivals. In this paper we consider five runway crossings to accommodate arriving aircraft.

C. Current Practice

In the course of operations the ground controller has many choices for routing aircraft from the apron (the area where the aircraft load and unload passengers) to the runway. However, to simplify routing, the set of possible choices has been limited to a set “playbook”, each “play” tells the pilot which taxiway to use and ultimately which

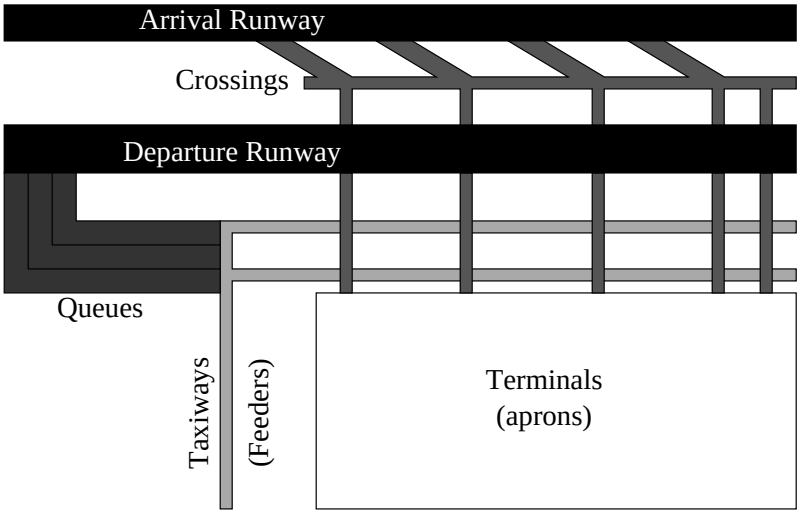

Figure 1: A Simplified View of a Queueing Area at the North-East Corner of the DFW Airport.

queue to enter. The plays consist of a list of taxiways the aircraft should take, and the queue the aircraft should enter. This paper assumes that the taxiway decisions have been made, and that the feeder the aircraft will use to approach the queueing area is given.

The ground controller doesn't have the capacity to deal with all the constraints efficiently as the aircraft reach the queueing area so they use shortcuts to speed up the decision-making process. For instance, if there are several aircraft that are subject to fix constraints then the controller might put them all into a single queue and let them wait their turn without regard to their ordering within the queue. This approach allows aircraft that are not subject to a fix constraint speedy access to the runway by keeping two queues open, but it could cause undue delay for certain fix-constrained aircraft.

D. Research Objectives

The goal of this research was to develop a series of algorithms to generate optimal departure sequences quickly enough to be implemented in a real-time simulation system.

E. Assumptions

This research involves several simplifying assumptions. The current practice of departure sequencing is assumed to reflect the First-in First-out (FIFO) ordering. Human controllers do deviate from the FIFO order, however no study has been done to characterize the sequences produced by human controllers. We assume that there is no upper-limit on the number of aircraft that can occupy a queue, crossing, or feeder. The optimal departure sequence would be affected by the differences in aircraft cruising speed, however this research does not consider the cruising speeds. This research assumes that aircraft cannot be reordered while in the feeders, however the geometry of airport can allow reordering.

F. Literature Review

There are several heuristics[2, 3, 4, 5] and optimal algorithms[6, 7, 8, 9] available for addressing aircraft scheduling problems in the literature. Most of the work related to the departure scheduling problem (DSP) in air traffic control has been in the area of scheduling aircraft landings. The constraints in problems involving landing aircraft are similar to the constraints in the DSP. The precedence (or ordering) constraints of the DSP addressed in this paper have a special structure where the departing aircraft are queued in the form of chains. These chain-like ordering constraints present in the DSP represent a simplified model of the physical layout of the runway queues

presented earlier. A set of arriving aircraft does not necessarily have this special structure.

Irrespective of whether an algorithm produces an optimal or a good approximation of the optimal solution, it is important to note that it would be useful to develop algorithms that can ultimately be used in a real-time simulation system. An exact algorithm produces optimal solutions but may have a running time that could make it infeasible in a real-time simulation. On the other hand, a heuristic could run fast but there are no guarantees on the solution quality. Optimal costs or tight lower bounds to the optimal costs are required to evaluate the quality of a heuristic.

It is important to understand that aircraft scheduling problems—such as the departure scheduling problem—differ from the classic Traveling Salesman Problem (TSP) with timing constraints or the single machine, job scheduling problem in the following way: if an aircraft, i, is departing after another aircraft, j, the departure time of aircraft i is not only dependent on the departure time of aircraft j, but also may depend on the departure time of aircraft departing before j. This could happen due to the additional separation constraints present for aircraft when they fly to a common departure fix. In the following discussion, a review of the existing literature related to the single runway, aircraft scheduling problem is presented.

Dear and Sherif were among the earliest to address the static and dynamic scheduling of landing aircraft.[2, 3] In static scheduling, a sequence is determined for a given set of aircraft. In dynamic scheduling, new aircraft are added continuously to the system and the schedules are updated frequently to include the new set of aircraft. Dear and Sherif introduced the concept of Constraint Position Shifting (CPS) as a feasible way to address the dynamic problem.[2] In this concept, a First Come First Served (FCFS) Sequence is initially generated based on the predicted landing times of all the aircraft. Then, an optimal sequence is generated such that no

aircraft can be shifted more than a given number of positions away from its original position in the FCFS sequence. For example, if the position of an aircraft in the FCFS sequence is 5 and the maximum number of shifts allowed is 1, then the aircraft in the optimal sequence can be in positions 4, 5, or 6. If CPS is not present, the position of an aircraft can be shifted several places for each update of the aircraft sequence. Therefore, by incorporating CPS while scheduling aircraft, one can eliminate these huge shifts in the positions of the aircraft. Heuristics were presented in Dear and Sherif to solve the aircraft scheduling problem with CPS.[3]

There are several other heuristics available for variants of the aircraft scheduling problems.[4, 5, 7, 10, 11, 12] Venkatarishnan et al.[4] presented a heuristic based on the dynamic programming approach by Psaraftis to solve the arrival scheduling problem with time window constraints.[13] Genetic algorithms are given in Abela et al.[5], and Ernst et al.[7] to solve a generlization of the arrival scheduling problem.

There are few ways in which optimal solutions can be obtained for aircraft scheduling problems. One way is to formulate the problem as an integer or mixed integer linear program (MILP) and solve the resulting program using any standard optimization software (CPLEX, GLPK).[6, 7, 14] This approach has a drawback, in the sense that the running times of the solvers could vary significantly[14] depending on a given instance of the problem. However, it is important to note that this approach can deal with several generalizations of the DSP. For example, it can readily deal with problems where the separation times do not follow the triangle-inequality.

There are four papers that are most relevant to this work. Three of the papers are from Atkin et al. who investigate the taxiway layout of the London Heathrow Airport by considering a complex structure of holding points.[10, 11, 12] The group goes on develop meta-heuristics (simulated annealing, tabu search) to improve traditional heuristic solutions to the departure scheduling problem. Brinton et al. ...

G. Contribution

The departure scheduling problem considered in this thesis has not been addressed in the literature. The following are the main contributions of this work:

1. construction and improvement heuristics for solving the departure scheduling problem while considering runway crossings from arriving aircraft,
2. lower bounds on the optimal solution using a mixed integer-linear program formulation, and
3. results from simulations showing sequence improvements in the direction of optimality over FIFO ordering.

CHAPTER II

DEPARTURE SCHEDULING WITH A SINGLE FEEDER

The initial work was essentially a proof of concept involving a simple swap, a feasibility check, and a check of the objective function. To facilitate the feasibility checking only a single feeder was considered. No results were generated from this algorithm, but the experience gained while developing and coding the algorithm was invaluable to the rest of the project.

A. Swap Procedure

Algorithm 1 was used to make the swaps in the preliminary work. Lines 1–3 are initialization, 9 and 10 are the swap. Lines 11–18 is where the feasibility and objective value checking takes place. The preliminary algorithm continues to make swaps until there is no improvement found in lines 11–18.

B. Order Feasibility

The number of queues needed to accommodate a sequence is the same as the minimum number of increasing disjoint subsets contained in the proposed take-off sequence. Given a proposed order and the number of available queues, the feasibility of an order can be found using Algorithm 2 (assuming the initial order is the natural order).

In Algorithm 2 lines 1 and 2 are initialization, gathering the proposed order, and setting the number of queues needed, n, to zero. The while loop on lines 3–12 will be executed once for every queue needed. The variable l holds the last aircraft that was moved to a queue, and it is re-initialized with each new queue. The loop on lines 5–10 executes for each aircraft that has not been moved to a queue, inside the loop the

```
1: I ← initial order (list)
2: improvement ← 1
3: oldObjective ← PRELIMOBJECTIVECHECK
4: while improvement = 1 do
5:    improvement ← 0
6:    for i ← 1 → |I| do
7:       for j ← 1 → |I| do
8:          P ← I
9:          P[i] ← I[j]
10:         P[j] ← I[i]
11:         if PRELIMORDERFEASIBLE then
12:            objecive ← PRELIMOBJECTIVECHECK
13:            if objective < oldObjective then
14:               I ← P
15:               oldObjective ← objective
16:               improvement ← 1
17:            end if
18:         end if
19:      end for
20:   end for
21: end while
```

Algorithm 1: Single Feeder Swap Algorithm

```
1: P ← proposed order (list)
2: n ← 0
3: while P ≠ ∅ do
4:    l ← 0
5:    for i ∈ P do
6:       if i > l then
7:          P ← P \ i
8:          l ← i
9:       end if
10:   end for
11:   n ← n + 1
12: end while
13: if n > MaxQueues then
14:   return false
15: else
16:   return true
17: end if
```

Algorithm 2: Single Feeder Order Feasibility Algorithm, PRELIMORDERFEASIBLE

current aircraft is tested to see if it falls after the aircraft previously moved (and thus is able to enter the queue). If it does fall after then it is removed from the proposed order (is put into the queue), and the variable l is updated. The variable n is checked against the number of available queues.

It is possible to adapt the order feasibility algorithm to execute faster by stopping the while loop once the available number of queues is reached. It is also possible to modify it to accept an arbitrary initial order. Figure 2 shows an execution of the algorithm for an order which requires three queues, and Figure 3 shows a four queue case.

Initial Order	1	2	3	4	5	6	7	8	9
Proposed Order	1	5	3	7	2	6	4	8	9
Queue 1	1	5		7				8	9
Queue 2			3			6			
Queue 3					2		4		
Queue 4									∅

Figure 2: Order Feasibility Checking Visualized, Three Queues Required

Initial Order	1	2	3	4	5	6	7	8	9
Proposed Order	1	8	6	7	2	4	3	5	9
Queue 1	1	8							9
Queue 2			6	7					
Queue 3					2	4		5	
Queue 4							3		
Queue 5									∅

Figure 3: Order Feasibility Checking Visualized, Four Queues Required

C. Objective Check

Apart from being feasible, the proposed order had to represent a better takeoff order than the incumbent, which means the objective function had to be lower. For the purposes of the preliminary algorithm—not in the interest of speed—the objective was re-tabulated for each feasible order. Finding the (throughput) objective value involves iterating through all the aircraft (in the order of the proposed take-offs), and tightening their leave times based on the required separation between aircraft types, as is shown in Algorithm 3. If the last aircraft's leave time is lower than that of the previous order's last leave time then the order is kept and the algorithm continues searching for an improvement.

```
P ← proposed order (list)
objective ← 0
for i ← 2 → |P| do
   objective ← objective + Separation(P[i − 1], P[i])
   if objective < Leave(P[i]) then
      objective ← Leave(P[i])
   end if
end for
return objective
```

Algorithm 3: Single Feeder Objective Checking Algorithm, prelimObjectiveCheck

CHAPTER III

GREEDY HEURISTIC

The greedy heuristic will generate a feasible take-off sequence given the initial conditions of a problem instance. To do this it chooses the "best" aircraft to send to the takeoff order from a pool of aircraft chosen from all the unscheduled aircraft. The pool of aircraft represents those that are available for takeoff: those at the head of a queue, the head of a crossing queue, or—if a queue is open—any aircraft in a feeder. If no aircraft meet the pooling criteria then the (un-departed) aircraft with the lowest early departure time is scheduled for take-off at its early departure time. Choosing from this pool ensures that the final take-off order is feasible. Figure 4 shows a few iterations of the greedy heuristic, and the pseudocode for the algorithm is given as Algorithm 4.

A. Required Separation

As was explained in Chapter I, there is a prescribed amount of time that is required to elapse between any two aircraft utilizing the runway (to take off). This amount of time is dependent on the leading and following aircrafts' type, as defined by the FAA.[15] The three main aircraft classes that use DFW are Large, Heavy, and the Boeing 757; the separation times required are shown in Table I.

Since this research also deals with aircraft crossing the runway it is necessary to expand the separation matrix to reflect the required crossing times.[15] Table II shows the revised separation matrix.

There is a special case where the separation matrix doesn't hold, and that is the "Simultaneous Crossing", wherein two arriving aircraft are allowed to cross the runway at the same time. At that point in time the runway is just another taxiway,

```
1: Q[i] ← Aircraft in ith queue (ordered list)
2: C[i] ← Aircraft in ith crossing queue (ordered list)
3: F[i] ← Aircraft in ith feeder (ordered list)
4: T ← ∅ {Takeoff order}
5: while Q ∪ C ∪ F ≠ ∅ do
6:    pool ← ∅ {Pool of aircraft to compare}
7:    emptyQueue ← false
8:    for i ∈ Q do
9:       if Q[i] = ∅ then
10:         emptyQueue ← i
11:      else
12:         pool ← pool ∪ HEAD(Q[i])
13:      end if
14:   end for
15:   if emptyQueue ≠ false then
16:      for i ∈ F do
17:         pool ← pool ∪ F[i]
18:      end for
19:   end if
20:   for i ∈ C do
21:      pool ← pool ∪ HEAD(C[i])
22:   end for
23:   incumbent ← null
24:   for i ∈ pool do
25:      incumbent ← COMPAREINCUMBENT(i, incumbent)
26:   end for
27:   if incumbent ∈ F then
28:      i ← incumbent[location]
29:      for j ∈ F[i] do
30:         if j = incumbent then
31:            break
32:         end if
33:         qIncumbent ← null
34:         for k ∈ Q \ emptyQueue do
35:            qIncumbent ← COMPAREQINCUMBENT(HEAD(Q[k], incumbent)
36:         end for
37:         POPAIRCRAFT(HEAD(F[i]), qIncumbent)
38:      end for
39:   end if
40:   POPAIRCRAFT(incumbent, T)
41: end while
```

Algorithm 4: Main function for the greedy algorithm

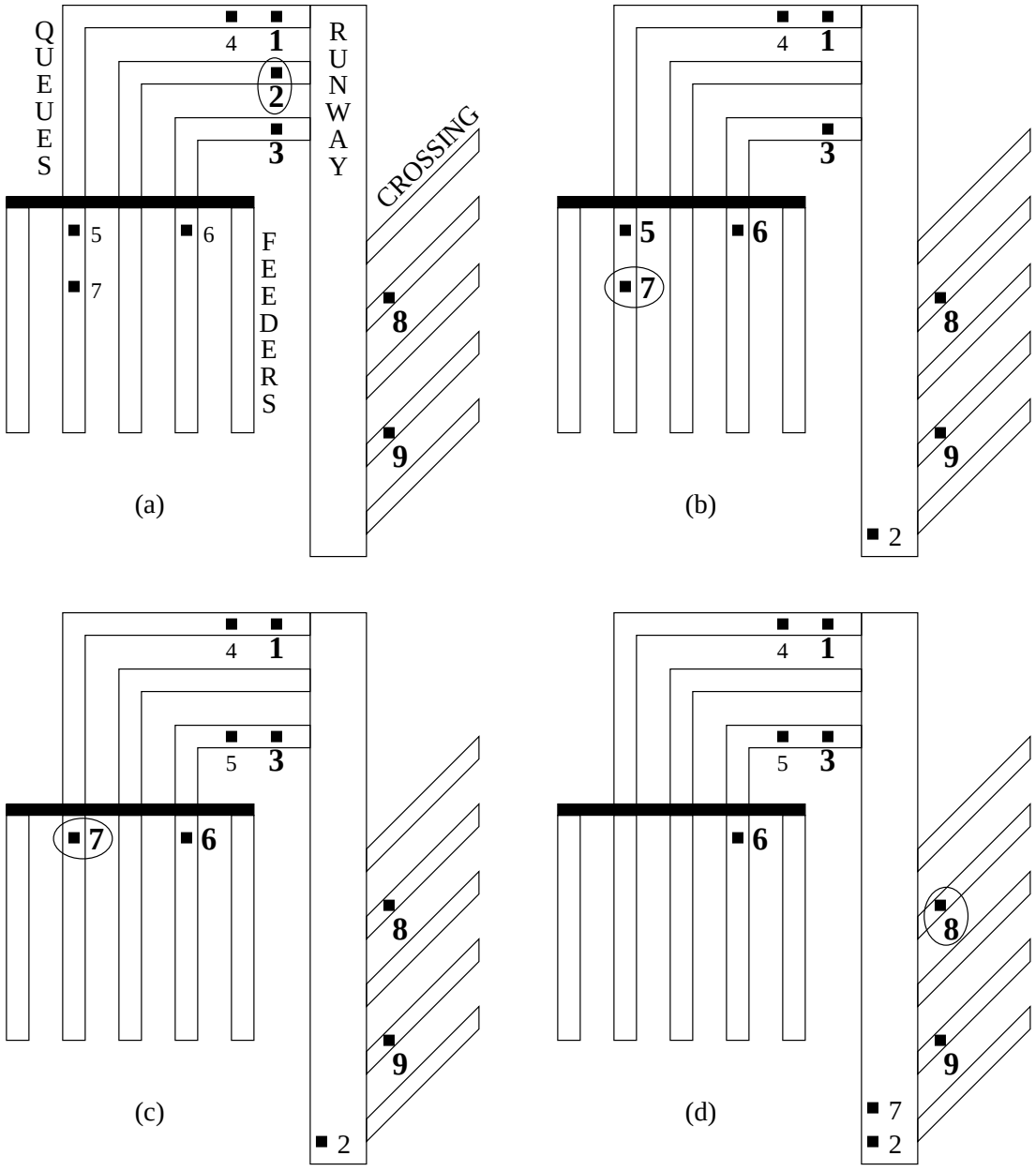

Figure 4: Iterating Through the Greedy Heuristic: The bold numbers represent aircraft that are in the pool, the circled number represents the aircraft selected to take off next. (a) Only the aircraft at the heads of the queues and crossings are in the pool, aircraft 2 takes off. (b) Only the aircraft not at the head of the queue or crossing (aircraft 4) is not in the pool, 7 is selected, but cannot move because aircraft 5 is in the way. (c) Aircraft 5 is moved to a queue, 7 is now free to move. (d) Aircraft 7 takes off, pool re-established.

Table I: Minimum Separation (in Seconds) Required Between Aircraft[15]

		Following		
	Type	Large	Heavy	B757
Leading	Large	55	75	55
	Heavy	110	100	110
	B757	90	75	60

Table II: Minimum Separation (in Seconds) Required Between Aircraft Including Crossings

		Following			
	Type	Cross	Large	Heavy	B757
Leading	Cross	10	40	40	40
	Large	55	55	75	55
	Heavy	55	110	100	110
	B757	55	90	75	60

so there are no restrictions other than the inline separation. That said, the separation between two crossing aircraft is only 10 seconds when the two crossing aircraft are using the same crossing point.

The other factor that limits an aircraft's leave time is the aforementioned fix-constraint. If two aircraft are supposed to go toward the same direction (or fix), then there could be an airspace constraint that requires them to stay a certain distance apart. So, the algorithm also checks the departing aircraft's fix to ensure that the fix separation is not violated.

B. Greedy Choice

Once the pool of potential take-offs has been determined the greedy choice can be made. The criteria for the choice depends on the chosen objective function, for the throughput objective the aircraft that is available for take-off the soonest should be

chosen. If delay (total, or max) is of interest then the choice needs to be based on the delay as well as the separation.

Should the best greedy choice happen to be an aircraft located in the feeder, then the algorithm must move all the aircraft in front of the chosen plane into a queue. The queue selection is based on the required separation between the aircraft to be moved and those already present in the queues, and—of course—the open queue is omitted from the selection.

C. Implementation

To facilitate the pool selection, a linked list representing each queue, feeder, and crossing is stored. The linked lists hold an identifying integer for each aircraft held therein, there is also a linked list representing the take-off order. Storing the aircraft in this way allows for initial conditions to be set easily, so it is possible to find an improved takeoff sequence at any point in your planning horizon. To prevent accounting errors, the function that moves an aircraft from one linked list to another can only move an aircraft from the head of one list to the tail of another. A global array of all aircraft is maintained which stores all the data pertaining to early times, leave times, aircraft type, fix, and location.

D. Objective Check

Checking the objective value is done after all the aircraft have been scheduled, the makespan of the departure sequence is equal to the departure time of the last aircraft in the sequence. For optimization based on delay it is necessary to iterate through each of the aircraft keeping summation of each aircraft's delay.

CHAPTER IV

2-INTERCHANGE HEURISTIC

The primary inspiration for this research was from the Ascheuer et al. paper dealing with the Asymmetric Traveling Salesman Problem with Time Windows (ATSP-TW). In the paper they outline an exact solution to the ATSP-TW using a branch and cut algorithm, they also present a series of heuristics they employed to solve the same problem.[16] Their approach was to strategically run their data through a large number of algorithms in order to continuously improve their solution. With each added heuristic they were able to get closer to the optimal solution, but since the algorithms were all efficient they did not significantly increase the overall running time (at least not to the level of the optimal solution runtime).[16] Ascheuer employed a total of 10 heuristics in several different configurations and in many of their test cases they were able to solve the problem to optimality in a fraction of the time required for the branch-and-cut algorithm.[16] The success of Ascheuer et al. was a motivating factor to tailor the 2-opt heuristic to the departure scheduling problem.

The k-interchange concept involves modifying a sequence (or traveling salesman tour) by replacing k edges with a different set of k edges to—hopefully—improve the objective. As Savelsbergh points out the 2-interchange can be quite fast in implementation, but as k grows larger the running time of the algorithm can suffer,[17] so the implementation in this research is currently limited to the 2-interchange. In a 2-interchange two links in the departure sequence are replaced thereby reversing the aircraft order between the interchanged nodes, Figure 5 shows a 2-interchange graphically.

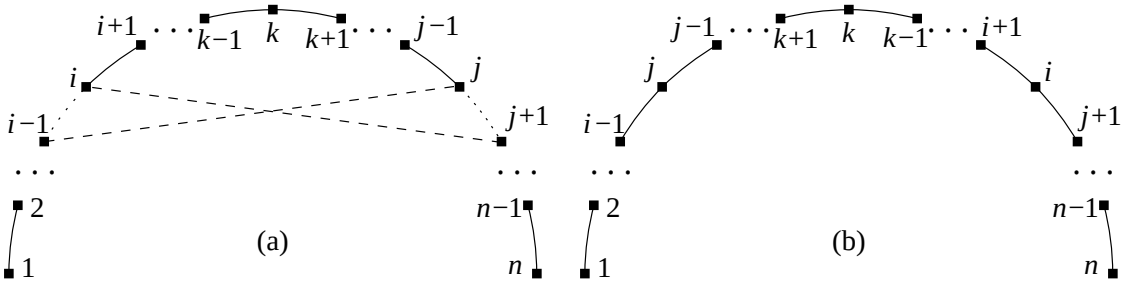

Figure 5: Example of a 2-Interchange: (a) The original order, the dotted lines represents the links that are to be broken, and the dashed lines show the new links that are to be forged. (b) The updated order, notice the reversal of the aircraft within the interchange.

A. Interchange Procedure

What the algorithm does is iterate through all the possible 2-interchanges accepting a change in the proposed order if the objective is improved and the order is feasible. In order to prevent the algorithm from slipping into the $O(n^3)$ complexity the swap window (or swap span) was limited, the span was set to five aircraft for the majority of the testing, this means that the scope of the subset order reversal was limited to five aircraft. This swap span was implemented in an attempt to maintain Dear and Sherif's idea of Constraint Position Shifting, to prevent an aircraft from being shifted too far from its original position.[2] In practice an aircraft can still move far from its original position in the takeoff order, however it will require several interchanges to do so.

B. Order Feasibility

Checking the order feasibility is a non-trivial part of this algorithm. Given an initial (feasible) order, $\{1, 2, 3, \ldots, n\}$, when a 2-interchange is performed on aircraft i and

```
1: I ← initial order (list)
2: P ← initial order (list) {This will hold the proposed order}
3: currObj ← objectiveCheck(I)
4: span ← swap span
5: for i ← 1 → |I| do
6:    for j ← i + 1 → i + span + 1 do
7:       for k ← 0 → j − i do
8:          P[i + k] ← I[j − k]
9:       end for
10:       if orderFeasible(P,i,j) then
11:          obj ← objectiveCheck(P)
12:          if obj < currObj then
13:             currObj ← obj
14:             I ← P
15:          end if
16:       end if
17:    end for
18: end for
```

Algorithm 5: The 2-opt Heuristic

j $(i < j)$ a proposed order, $\{1, 2, 3, \ldots, i-1, j, j-1, \ldots, i+1, i, j+1, \ldots, n\}$, is formed (like Figure 5) and its feasibility needs to be confirmed. The feasibility check of the proposed order can be broken down into four cases, all but the first case ignore arriving aircraft waiting in the crossing queue; adding crossings only makes the remaining cases trivially more difficult so long as the instance passes the first case. For an aircraft to be considered "in the interchange" it must fall between the ith and jth aircraft, and thus the order of its neighbors are reversed in the proposed order.

Case 1 At least two aircraft in the interchange belong to the same queue. This case is infeasible since aircraft in a specified queue must take off in the same order as they enter the queue (the same holds for crossings).

Case 2 All aircraft in the interchange belong to different queues, and no aircraft are in feeders. This case is feasible since aircraft in different queues can depart in any order.

Case 3 None of the interchanged aircraft, or aircraft after the interchange, are in the queues, all of them are in feeders. To check the feasibility of the proposed order a revised initial order needs to be generated This revised order is generated by iterating through the proposed order locating each aircraft in its feeder and pulling it (as well as any aircraft in front of it in the same feeder that hasn't previously been pulled forward) forward to the revised order. Figure 6 shows an example of generating a revised initial order. The revised order is a representation of the multiple feeders in a single feeder. Doing the increasing subsequences test against the revised initial order will give a confirmation of the feasibility of the proposed sequence. It should be noted that, unlike the feasibility algorithm outlined in Chapter II, the increasing subsequences are not

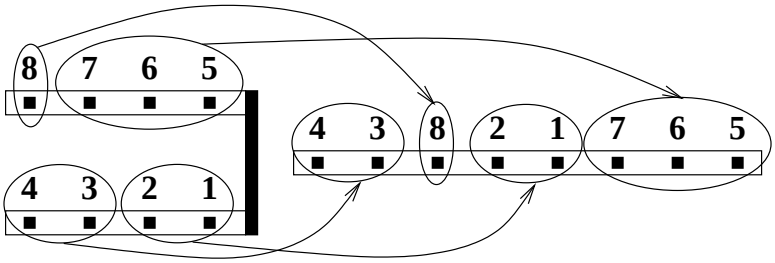

Figure 6: Generating a Revised Initial Order: Given a proposed departure sequence, $\{7, 5, 2, 1, 8, 4, 3, 6\}$. Aircraft 7, being the first to depart, is pulled forward along with the aircraft in front of it (5 and 6). Since 5 has already been pulled forward it is skipped, 2 is the next aircraft to be pulled forward (1 comes with it). Then 8, then 4 (with it 3). Performing the increasing subsequences test against the revised order shows that the proposed order is feasible.

generated from the order of the natural numbers but instead from the revised initial order from the feeder.

Case 4 There is at least one aircraft in or after the interchange that is in a queue. The feasibility of the feeder orders which contain an aircraft in the interchange needs to be checked as was done in Case 3. However, a subset of the aircraft representing the occupied queues must be added to the beginning of the revised feeder order before checking its feasibility. This representative subset should include the last aircraft from each queue that is proposed to take off after aircraft $i-1$, and the subset's order should be reversed from that of the proposed takeoff order.

```
1: P ← Order to be tested
2: start ← i Start of interchange (index)
3: end ← j End of interchange (index)
4: inQueue[:] ← 0 {Track the number of aircraft in each of the queues}
5: inFeeder[:] ← 0
6: inCross[:] ← 0
7: for i ∈ P[start : end] do
8:    if i[locale] = QUEUE then
9:       inQueue[i[location]] + +
10:      if inQueue[i[location]] > 1 then
11:         return false{Case 1}
12:      end if
13:   else if i[locale] = FEEDER then
14:      inFeeder[i[location]] + +
15:   else {i is in a crossing queue}
16:      inCross[i[location]] + +
17:      if inCross[i[location]] > 1 then
18:         return false{Case 1}
19:      end if
20:   end if
21: end for
22: for i ∈ P[end + 1 : |P|] do
23:   if i[locale] = QUEUE then
24:      inQueue[i[location]] + +
25:   end if
26: end for
27: interference ← ∅
28: if ∑_i inQueue[i] > 1 ∧ ∑_i inFeeder[i] > 1 then
29:   usedQueue[:] ← ∅
30:   for i ← |P| → start do
31:      if P[i][locale] = QUEUE ∧ P[i][location] ∉ usedQueue then
32:         usedQueue ← usedQueue ∪ P[i][location]
33:         interference ← interference ∪ P[i]
34:      end if
35:   end for
36: end if
37: {Continued in Algorithm 7}
```

Algorithm 6: Order Feasibility Checking for 2-opt Heuristic, ORDERFEASIBLE

```
1: P' ← P[start : end] {Copy the proposed order}
2: I' ← interference
3: while P' ≠ ∅ do
4:   if P'[1][locale] = FEEDER then
5:     for i ← 1 → |P| do
6:       if P[i][locale] = FEEDER ∧ P[i][location] = P'[1][location] then
7:         I' ← I' ∪ P[i]
8:         if P[i] = P'[1] then
9:           P' ← P' \ P[i]
10:           break
11:         else if P[i] ∈ P' then
12:           P' ← P' \ P[i]
13:         end if
14:       end if
15:     end for
16:   else
17:     P' ← P' \ P'[1] {Aircraft not in feeders are of no concern at this point.}
18:   end if
19: end while
20: count ← 0
21: testOrder ← ∅
22: for j ← 1 → |P| do
23:   if (P[j][locale] = FEEDER ∧ P[j][location] = i) ∨ P[j][locale] = QUEUE then
24:     for k ∈ P do
25:       if P[j] = k then
26:         testOrder ← testOrder ∪ count
27:         break
28:       else
29:         count ← count + 1
30:       end if
31:     end for
32:   end if
33: end for
34: if ¬PRELIMORDERFEASIBLE(testOrder) then
35:   return false
36: end if
37: return true{All tests passed}
```

Algorithm 7: Continuation of the Order Feasibility Checking Algorithm

C. Objective Check

Before accepting a proposed order, its objective function value must be confirmed as lower than the previous objective. As the objective function is checked, the algorithm stores the updated leave times for the aircraft within the swap, it also re-orders the fix links so the fix-constraints are maintained. Once the leave times for the aircraft in the swap are known and stored the objective check begins.

Using the leave times for the aircraft within the swap, the leave times for all aircraft after the swap are calculated and the pertinent data are used for objective comparison. This ensures that the objective is improved on each accepted iteration. Algorithm 8 shows generally how the objective checking is done.

```
1: P ← Order to be tested
2: prevAC ← null {Previous aircraft to utilize the runway (can be an arrival)}
3: prevTO ← null {Previous aircraft to take off}
4: prevAtFix[:] ← null {Previous aircraft at each fix}
5: usedCross ← ∅
6: cumDelay ← 0 {Cumulative Delay}
7: for i ∈ P do
8:    if prevAC = null then
9:       i[leave] ← i[early]
10:      prevAC ← i
11:      if i[locale] ≠ CROSSING then
12:         prevAtFix[i[fix]] ← i
13:      end if
14:      continue
15:   end if
16:   if i[locale] = prevAC[locale] = CROSSING ∧ i[location] ∉ crossUsed then
17:      i[leave] ← max{prevAC[leave], i[early]}
18:      cumDelay ← cumDelay + i[leave] − i[early]
19:      usedCross ← crossUsed ∪ i[location]
20:   else
21:      usedCross ← ∅
22:      leave1 ← prevAC[leave] + sep[prevAC][i]
23:      leave2 ← prevTO[leave] + sep[prevTO][i]
24:      if i[locale] ≠ CROSSING then
25:         leave3 ← prevAtFix[i[fix]][leave] + fixSep[i[fix]] − prevAC[leave]
26:         prevAtFix[i[fix]] ← i
27:         prevTO ← i
28:      else
29:         leave3 ← 0
30:      end if
31:      i[leave] ← max{i[early], leave1, leave2, leave3}
32:      cumDelay ← cumDelay + i[leave] − i[early]
33:   end if
34:   prevAC ← i
35: end for
36: if OBJECTIVE = TOTALDELAY then
37:    return cumDelay
38: end if
39: if OBJECTIVE = THROUGHPUT then
40:    return prevAC[leave]
41: end if
```

Algorithm 8: Objective Checking for the 2-opt heuristic, OBJECTIVECHECK

CHAPTER V

LOWER BOUNDS USING A MIXED INTEGER LINEAR PROGRAM

In order to benchmark the performance of the heuristics it was necessary to find the optimal solution to the problem, or at least a bound on the solution. The Mixed Integer Linear Program (MILP) formulation and description follows. The variables used are described below.

A The set of aircraft to be scheduled.

K The set of queues and crossing queues.

a_i The earliest possible take-off time for aircraft i, assuming the aircraft's taxi to the runway is unencumbered.

t_i Decision variable. The actual (calculated) take-off time for aircraft i.

c_i The class (type) of aircraft i, the class of an arriving aircraft should indicate that it is an arrival.

d_i The departure flag for aircraft i, if aircraft i is departing then $d_i = 1$, if aircraft i is an arrival then $d_i = 0$.

f_i The fix that aircraft i is departing along. The departure fix for an arrival should be null.

S_{ij} The required separation between an aircraft of class i (leading) and class j (following).

F_i The required separation along fix i.

z_{ij} Decision variable. Take-off ordering, if aircraft i precedes aircraft j in the take-off order then $z_{ij} = 1$, otherwise $z_{ij} = 0$.

α_{ij} Queue, feeder, or crossing order. If aircraft i and j are in the same queue, feeder, or crossing, and aircraft i precedes aircraft j then $\alpha_{ij} = 1$, otherwise $\alpha_{ij} = 0$.

y_i^k Decision variable. The queueing of aircraft i, if aircraft i enters (or begins in) queue k then $y_i^k = 1$.

Now for the problem constraints.

$$z_{ij} + z_{ji} = 1 \qquad \forall i, j \in A\colon\ i \neq j \tag{5.1}$$

$$a_i \leq t_i \qquad \forall i \in A \tag{5.2}$$

$$\sum_{k \in K} y_i^k = 1 \qquad \forall i \in A \tag{5.3}$$

$$z_{ij} \geq \alpha_{ij} - (2 - y_i^k - y_j^k) \qquad \forall i, j \in A,\ \forall k \in K\colon\ i \neq j \tag{5.4}$$

$$(z_{ij} - 1)M \leq t_j - t_i - S_{ij} \qquad \forall i, j \in A\colon\ i \neq j \tag{5.5}$$

$$(z_{ij} - 1)M \leq t_j - t_i - F_{f_i} \qquad \forall i, j \in A\colon\ i \neq j, f_i = f_j \tag{5.6}$$

$$z_{ij} \in \{0, 1\} \qquad \forall i, j \in A\colon\ i \neq j \tag{5.7}$$

$$y_i^k \in \{0, 1\} \qquad \forall i \in A,\ \forall k \in K \tag{5.8}$$

Equation 5.1 exists to prevent ambiguity in the take-off order, it prevents an aircraft from being both in front of and behind another aircraft. Equation 5.2 prevents an aircraft from being scheduled before it is available for departure. The constraint given in Equation 5.3 ensures that an aircraft only belong to a single queue or crossing. Equation 5.4 enforces proper re-ordering when moving from a feeder to queue or a queue to the take-off order. This constraint has only been proven to hold for the single feeder case, thus any results presented for the multiple feeder case represent a lower bound on the optimal solution.

Proper separation between aircraft (based on class) is ensured by Equation 5.5, this constraint uses a large integer M to overcome a non-linearity in the desired constraint given by Equation 5.9.

$$z_{ij}(t_j - t_i - S_{ij}) \geq 0 \qquad \forall i, j \in A\colon\ i \neq j \tag{5.9}$$

To accommodate simultaneous crossing Equation 5.5 must be further modified to Equation 5.10, which relaxes the separation constraint if both the lead and follow aircraft are arrivals and do not belong to the same crossing.

$$(z_{ij}-1)M \leq \begin{cases} t_j - t_i, & \alpha_{ij} \neq 1,\ \alpha_{ji} \neq 1,\ d_i = d_j = 0; \\ t_j - t_i - S_{ij}, & \text{otherwise.} \end{cases} \qquad \forall i, j \in A:\ i \neq j \tag{5.10}$$

The fix constraints are dealt with by Equation 5.6, if two aircraft have the same departure fix then they must be separated by the prescribed amount. Equations 5.7 and 5.8 ensure that the associated decision variables remain binary—of course, to solve the problem these constraints are relaxed to $\cdots \in [0, 1]$.

For the purposes of initialization the initial conditions in the queues can be set by defining another input to the problem: Q_i^k, wherein $Q_i^k = 1$ if aircraft i is initially in queue k. Further the constraint given in Equation 5.11 needs to be added to the formulation.

$$y_i^k \geq Q_i^k \qquad \forall i \in A,\ \forall k \in K \tag{5.11}$$

A. Objective Functions

Depending on what objective value is sought one of the following arrangements should be used. For the throughput objective a constraint should be added:

$$Z \geq t_i \qquad \forall i \in A,$$

and Z should be minimized. This is an adaptation of the standard min-max objective. The max delay objective is another adaptation of the min max, this time D should

be minimized and the following constraint added:

$$D \geq t_i - a_i \qquad \forall i \in A.$$

The total delay doesn't require an added constraint, instead set the following as the objective:

$$\min \sum_{i \in A} t_i - a_i.$$

CHAPTER VI

CONCLUSIONS

A. Testing Schema

To test and benchmark the algorithm a series of tests were run on the many different facets of the code, and—when applicable—the optimal solution was found using the MILP formulation. For the purposes of testing, several subroutines were written for file input/output, data generation, and to find the First-in First-out (FIFO) objective value. The code will accept (through the command line) a file holding the pertinent aircraft data—it can also output data in this format—as well as outputting data in the format required by the MILP solver (OPL/CPLEX). For testing, a case is randomly generated (as described below) and the data stored to a file in both formats, then the following tests were run and the objective values recorded:

1. First-in First-out discipline,
2. Greedy heuristic,
3. 2-interchange heuristic, and
4. 2-interchange using the greedy order as initial feasible order.

Finally, if possible the optimal solution was found using the MILP, due to time restrictions only cases with fewer than 18 aircraft were solved to optimality. All the odd numbers of aircraft from 5–45 were tested with 50 cases each.

B. Data Generation

In order for the tests to be meaningful the aircraft had to be attempting to leave faster than than the runway could handle them[1]. The aircraft type is defined randomly (even probability) except for aircraft one, which is automatically defined as large. The first aircraft's leave time is set as zero, and the other aircraft's leave times are randomly chosen from a window which starts from the previous aircraft's leave time. The window length is based on a the desired number of departures per hour. The locale (queue, feeder, or crossing) of each aircraft is randomly assigned, but only aircraft leaving before a specified time (based on the number of aircraft) can enter the queue. Once the locale is determine the location within the locale can be set (randomly), the location corresponds to which of the feeders, queues, or crossings the aircraft is in. The fixes are also randomly generated with even probability.

C. Computational Results

Figure 7 shows the average running times for the various algorithms for the specified quantities of aircraft. The plot may not be clear, but it is clear that the run time hovers around 4 ms. As the number of aircraft approaches 50 the runtime increases, but only slightly, this is probably due to the fact that a large portion of the code is the overhead required to make the calculations: the memory allocation, and data-structure initialization. The average runtimes for MILP formulation are given in Figure 8, note that the dependent axis is given in seconds. The solution is reasonably fast for fewer than 15 aircraft, but entirely too slow for greater than 17 aircraft (the 17 aircraft cases average to 76.3 seconds, the point was omitted for clarity).

---

[1]Otherwise there would be a significant number of no-delay aircraft, thus little room for optimization.

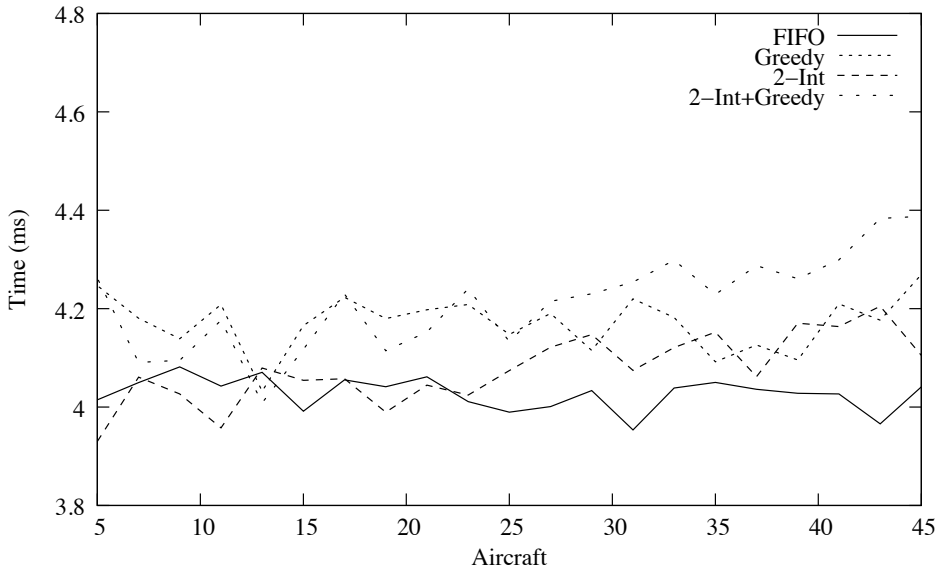

Figure 7: Average Run Times for the Various Heuristics

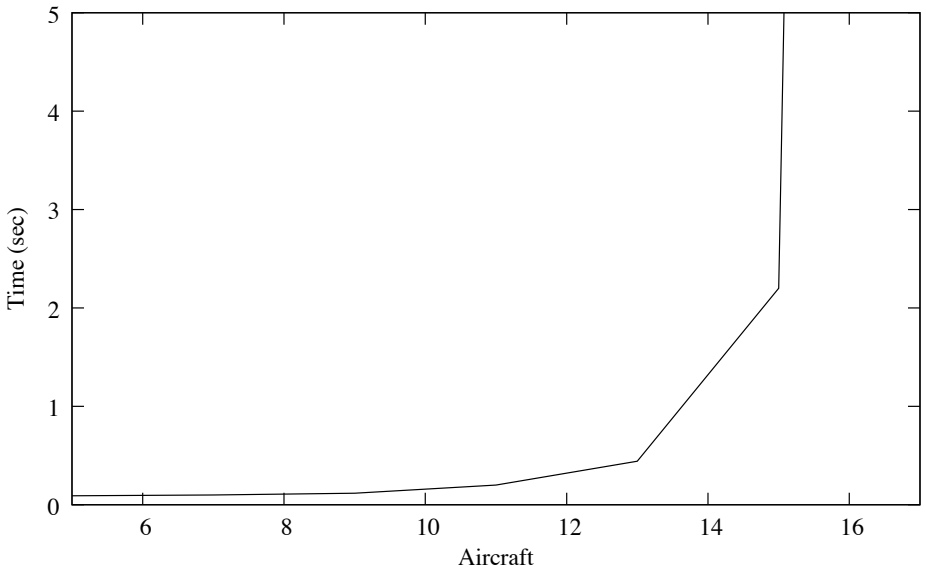

Figure 8: Average Optimal Solution Times

The solution quality is normalized against both the optimal solution and first-in first-out, Figure 9 shows the percent objective improvement of the various heuristics over the first-in first-out ordering. Twelve to fifteen percent improvement is possible from the most complicated algorithm. The swap algorithm alone yields a better result over the greedy algorithm. It should be noted that human operators do not necessarily maintain first-in first-out ordering, so the improvements shown are not necessarily realistic with regard to current practice. No study has been done to characterize the quality of sequences generated by human operators. Figure 10 shows the percent improvement of the optimal solution over the heuristics tested, once again the most complicated yields the best results, between seven and fifteen percent worse than the optimal solution. The swap alone gets within fifteen to twenty percent of the optimal solution.

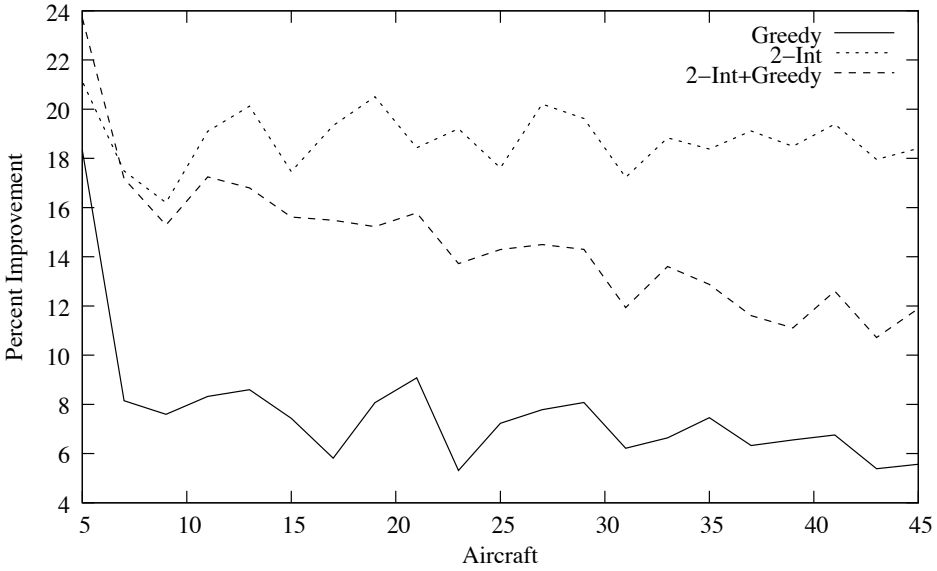

Figure 9: Average Percent Objective Improvement of the Various Heuristics Over the First-In First-out Order

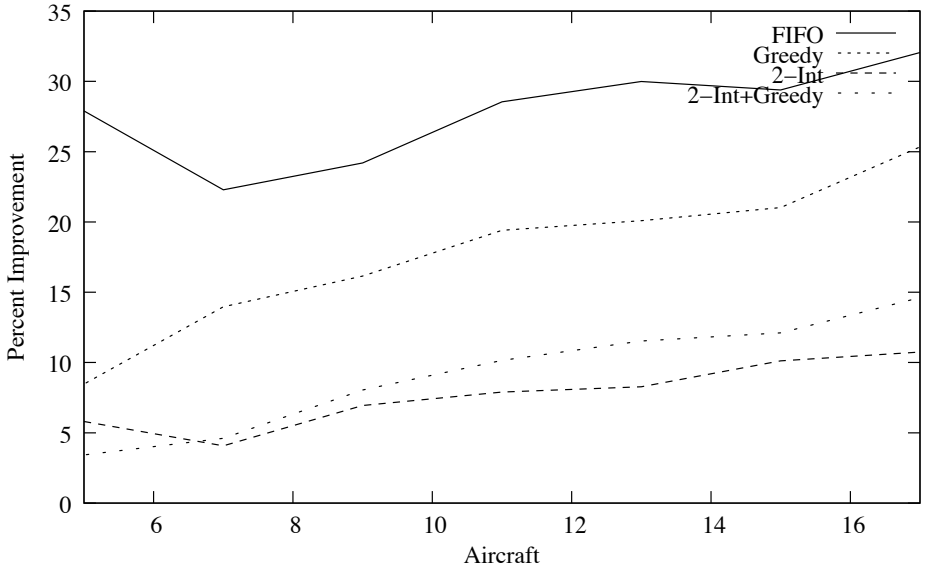

Figure 10: Average Percent Objective Improvement of the Optimal Solution Over the Various Heuristics

REFERENCES

[1] Joint Economic Committee, "Your flight has been delayed again," Tech. Rep., United States Senate, May 2008.

[2] R. G. Dear and Y. S. Sherif, "The dynamic scheduling of aircraft in high density terminal areas," *Microelectronics and Reliability*, vol. 29, no. 5, pp. 743–749, 1989.

[3] R. G. Dear and Y. S. Sherif, "An algorithm for computer assisted sequencing and scheduling of terminal area operations," *Transportation Research*, vol. 25, no. 25, pp. 129–139, 1991.

[4] C. S. Venkatakrishnan, A. Barnett, and A. R. Odoni, "Landings at logan airport: Describing and increasing airport capacity," *TranSci*, vol. 27, no. 3, pp. 211–227, 1993.

[5] J. Abela, D. Abramson, M. Krishnamoorthy, A. De Silva, and G. Mills, "Computing optimal schedules for landing aircraft," in *Proceedings of the 12th National ASOR Conference*, Adelaide, Australia, 1993, pp. 71–90.

[6] J. E. Beasley, M. Krishnamoorthy, Y. M. Sharaiha, and D. Abramson, "Scheduling aircraft landings—the static case," *TranSci*, vol. 34, no. 2, pp. 180–197, 2000.

[7] A. T. Ernst, M. Krishnamoorthy, and R. H. Storer, "Heuristic and exact algorithms for scheduling aircraft landings," *Networks*, vol. 34, no. 3, pp. 229–241, September 1999.

[8] H. Balakrishnan and B. Chandran, "Scheduling aircraft landings under constrained position shifting," in *Navigation and Control Conference*, Keystone, Colorado, August 2006, AIAA Guidance.

[9] H. Lee and H. Balakrishnan, "Fuel cost, delay and throughput tradeoffs in runway scheduling," in *Proceedings of the American Control Conference*, Seattle, Washington, June 2008, pp. 2449–2454.

[10] J. A. D. Atkin, E. K. Burke, J. S. Greenwood, and D. Reeson, *A Metaheuristic Approach to Aircraft Departure Scheduling at London Heathrow Airport*, vol. 600 of *Lecture Notes in Economics and Mathematical Systems*, pp. 235–252, Springer, San Diego, California, 2008.

[11] J. A. D. Atkin, E. K. Burke, J. S. Greenwood, and D. Reeson, "Hybrid metaheuristics to aid runway scheduling at london heathrow airport," *TranSci*, vol. 41, no. 1, pp. 90–106, 2007.

[12] J. Atkin, E. Burke, J. Greenwood, and D. Reeson, "An examination of take-off scheduling constraints at london heathrow airport," in *Electronic Proceedings of the 10th International Conference on Computer-Aided Scheduling of Public Transport*, Leeds, UK, June 2006.

[13] H. N. Psaraftis, "A dynamic programming approach for sequencing groups of identical jobs," *Operations Research*, vol. 28, no. 6, pp. 1347–1359, 1980.

[14] T. Fahle, R. Feldmann, S. Götz, S. Grothklags, and B. Monien, "The aircraft sequencing problem," in *Computer Science in Perspective: Essays Dedicated to Thomas Ottmann*, pp. 152-166. New York: SpringerVerlag, 2003.

[15] H. Balakrishnan and Y. Jung, "A framework for coordinated surface operation planning at Dallas-Fort Worth International Airport," in *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, Hilton Head, NC, August 2007.

[16] N. Ascheuer, M. Fischetti, and M. Grötschel, "Solving the asymmetric travelling salesman problem with time windows by branch-and-cut," *Mathematical Programming, Series A*, vol. 90, no. 3, pp. 475–506, May 2001.

[17] M. W. P. Savelsbergh, "Local search in routing problems with time windows," *Annals of Operations Research*, vol. 4, no. 1, pp. 285–305, December 1985.

VITA

Paul William Stiverson received his Bachelor's degree in mechanical engineering from Texas A&M University in 2006. He promptly entered graduate school at the same institution, and took a position as a Teaching Assistant for Dr. Dara Childs in Dynamics and Vibrations. His love for teaching soon spread and he took up tutoring other courses, eventually composing and typesetting a full note set for Dynamic Systems and Controls. In the summer of 2008 he was invited to work at the NASA Ames Research Center in Moffett Field, California by the University Affiliated Research Center. At Ames he began the research that would become his Master's thesis work, and upon his return to Texas A&M he resumed his teaching assistantship. In the spring of 2009 he was awarded funding by the Education Associates Program which provides scholarly funding for NASA's student researchers.

Paul may be reached by writing to Paul Stiverson, Texas A&M University—Mechanical Engineering, Mail Stop 3123, College Station TX 77845.