Scheduling Aircraft Landings

The Dynamic Case

Master Thesis

April 2007

Supervisor: Jens Clausen

Department of Informatic and Mathematical Modelling

Technical University of Denmark

Erling Veidal, s042875

Abstract

This Master Theses is about solving the aircraft landing problem dynamically. Given an original landing schedule for the incoming aircraft this schedule are rescheduled whenever an aircraft arrives at the inflight for the runway early or late compared to the original scheduled landing time. Three different approaches for dealing with early aircraft and four different solutions dealing with late aircraft are presented, implemented and tested. For rescheduling a part of the original schedule a population heuristic is created. Based on the results, the best approach for early aircraft is to deal with these as late aircraft and hence recalculate a part of the original schedule and insert the early aircraft as soon as possible. The different late approaches all gave the same results which was not expected, this is probarbly due to the fact that the time window for all the aircraft are so tight that the heursitics allways finds the same solution. The apperance time is the key factor in the rescheduling and will dictate the solution. When comparing the results to a FCFS (First-Come-First-Served) scheme that calculates the landing time according to the apperance time of the aircraft and takes the seperation constraints into account, the different approaches tested finds the same solution as the FCFS scheme. The different dynamic approaches have been tested on files containing from 10 to 100 aircraft.

Acknowledgement

I would like to thank the following persons for making this theses possible.

- My supervisor Jens Clausen, for opening my eyes for operational research and his help and support during this project.
- Christian Skaarup Hansen, for the discussions we had in the office and his solution ideas.
- Andreas Nauta Pedersen and Bjarne Poulsen for their reviews.

Finally I would like to thank Sanne for her patience and love, especially in the last days before I handed in this report.

Lyngby, april 2007
Erling Veidal

Contents

Chapter 1

Introduction

Ever since commercial aircraft became available in the western world in the late 1960's and early 1970's, there has been a dramatic increase in the air traffic. This is now at a point were the bottlenecks for increasing the air traffic further no longer are the number of planes available, but the capacities at the airports. Several of the busiest airports in the world have reached their maximum size, making it impossible to extend the airport with more runways due to environmental, political, noise or space considerations. In these airports it is necessary to maximize the use of their existing runways, to handle as many planes as possible. As an example, Heathrow-London and Kastrup-Copenhagen have only two runways, one used for takeoffs and one used for landings. These runways are changeable and are being switched according to wind conditions. The air-traffic-controllers typically schedule the incoming aircrafts in a first-come-first-served (FCFS) approach, giving a useable scheme for scheduling the landings but not maximizing the throughput at the runway. To some extent programs for helping in the scheduling are being used. If these programs can schedule the landings in an order different than FCFS that increases the number of landings with one per hour, the potential number of landings at that airport would increase with 8766 each year and thereby increasing the possible profit at the airport.

1.1 The problem

Scheduling aircrafts landings is not as easy as it might seem. There are several constraints that need to be fulfilled. These will be presented in the following sections.

1.1.1 Landing time

Logically, there is an earliest time when the aircraft can land, namely if it continues at its maximum speed towards the runway. Similarly there is a latest landing time, since an aircraft eventually will run out of fuel, and therefore cannot continue circling around the airport. These times defines the time window in which each aircraft must be landed. There also exists a target time for each aircraft. The time it was supposed to land according to its flight-schedule, or, if economical considerations have a high factor, the time it arrives if it continues at the aircrafts cruising speed until landing and thereby saving fuel. When scheduling aircraft landing each aircraft are given a buffer that gives them the possibility to arrive at their target time even when suffering delays from departure. Because of this buffer it is not unusual that aircraft arrive somewhat earlier than their target landing time. When dealing with the static case the earliest, latest and target landing time is based on some demands from the airlines so that they can maximize the usage of their aircraft in the planning on which aircraft to use on the different routes they operate.

1.1.2 Separation

An aircraft generate turbulence when flying; a larger aircraft generates more turbulence than a smaller aircraft. This generated turbulence can cause problems for other aircraft and in the worst case damage these. There is a parallel between the size of an aircraft and the turbulence it can endure. Large aircrafts can endure more turbulence than smaller aircrafts. Because of this turbulence there are separations constrains between aircraft that must be fulfilled for securing safe landings and flight. The constraints that should be taken into considerations due to legislations vary from case to case, depending on the involved aircraft.

1.1.3 Other constraints

Weather also has a great impact on how it is possible to schedule the aircraft. Wind conditions can cause landing to be cancelled, especially for smaller aircraft. Fog can cause visibility to be so low that it is not considered safe to let the planes land at the intervals they normally would. Lightening, heavy rain or snow/ice can also cause problem for the aircrafts. Finally an aircraft landing at one runway make some turbulence that impacts the aircraft landing at other runways. These cases are not considered in the existing literature for the aircraft landing problem and will not be discussed further in this report.

1.2 The static case

The calculation in the static case of scheduling aircraft landings is used for scheduling the optimal queue for aircraft waiting to land at one or more runways. This is typically done in the planning of when the different aircraft should land to maximize the usage of the runways or minimizing the difference between the target landing times and the actual landing time. Since the calculation is done before any actual aircraft in the calculation is near the airport the constraints on computational time are weak and since the number of aircraft is fixed, it is possible to find an optimal solution. Most of the articles describing the aircraft landing problem deals with the static case. Typically a sequence of landings scheduled by an air-controller are analysed, and then the authors search for a better solution based on some heuristics. In some of the articles there are also a strictly theoretical solution based on randomly generated aircraft and a fixed number of runways on which they can land. This latter approach is mostly used to compare the different heuristics or other methods for finding the optimal schedule.

1.3 The dynamic case

While the static case primarily is concerned with the planning of landings, the dynamic case optimizes the final adjustment to the scheduling of the incoming aircrafts.
In the dynamical case aircrafts do not always arrive in the order they were planed or at the expected time. This may be due to delays from taking off later than planed, weather conditions or other circumstances changing the time at which the aircraft arrives. Hence it is necessary to make some rescheduling of the aircraft. An approach might be to wait until an aircraft is inside the range of the airport's control tower radar and then recalculate the order in which the aircraft should land.

Case: The current aircraft waiting to be landed are scheduled. When a new aircraft arrives this schedule needs to be recalculated in order to give the best results. Several considerations need to be made;

- Should an aircraft be guaranteed that it does not change sequence if it is about to land?
  - If this is the case; for how many of the aircraft currently planed to land next should this apply?
- Should there be penalty function for changing the aircraft with larger penalties for the aircraft that are in the start of the queue rather than the ones at the end of the queue? The goal then becomes to minimize this penalty.
- Should the time window for each aircraft narrow if it is about to land?
- Is there a maximum number of aircrafts that may be rearranged?

Solving the dynamic problem is similar to solving the static case, but with some additional considerations about when to reschedule the sequence of the aircrafts, and which aircraft to reschedule.

1.4 Metaheuristics

In some cases, it is not possible to find the optimal solution to a specific problem. This could be the fact when the time calculating the optimal solution is unreachable in the specific problem. Heuristics are procedures that are likely to find a good feasible solution, but do not guarantee that it has found the optimal solution. Since there are no constraints saying that the optimal solution must be found, the heuristics proceeds with its search when one solution is found. At some criteria the heuristics stops, compares the found solutions and selects the best solution. Heuristics in general needs a lot of work for being able to solve a specific problem. For making heuristics more useable, some standard models are developed called metaheuristics. Metaheuristics are heuristics with a general structure, and with strategy guidelines for specialising for the problem at hand. Some of the most popular metaheuristics will be presented and shortly explained in the following. Each of the metaheurstics described are based on the information in [1].

1.4.1 Tabu search

Tabu search uses a local search approach. It begins by finding a local optimum, and then continues by allowing non-improvingly solutions. For preventing circles in the search, a tabu list is made containing previous visited solutions. Therefore, before moving to a new solution, the tabu list is first checked to see if that specific solution already has been visited. This could typically be implemented in such a way that moving to a neighbourhood solution the opposite movement is added to the tabu list and hence guaranteeing that the search will not visit the solution from which it arrived.
In general, the tabu search starts with an initial solution, and then searches the neighbourhood of solutions to see if there are any better solutions. If this is the case; the best of the neighbourhood solutions are selected as the new initial solution and the old initial solution are added to the tabu list. If there are no neighbourhood solutions that are better than the initial solution, the initial solution is saved as the best solution and then the best of the allowable solutions is chosen as the new initial solution. Since the size of the tabu list cannot be infinite, it normally has a fixed size. Therefore there need to be a procedure of replacing solutions in the tabu list. The easiest way is by implementing a first-in first-out scheme, which also provides flexibility since the search is not constrained around the best solutions so far. The stop criteria for the tabu search may be a fixed number of iterations, a fixed amount of CPU time or a fixed number of iterations without improving the best solution.
Even though tabu search is simple in its general form, there are still some

decisions that need to be made before it can be used to solve a specific problem:

- Which local search procedure to use.
- How to define the neighbourhood of solutions.
- Which tabu move (solution) should be added to the tabu list.
- The size of the tabu list.
- The stop criteria.

1.4.2 Simulated annealing

Simulated annealing is a metaheuristics that, just like tabu search, allows the search process to escape from a local optimum. In simulated annealing, a better neighbourhood solution is always accepted and a worse neighbour solution is accepted based on a probability function. The probability function is e^x, where x=$\frac{Z_n - Z_c}{T}$, Z_c is the objective function value of the current solution and Z_n is the objective function value of the next candidate to the trial solution. T is a temperature parameter that measures the tendency to accept the current candidate as the next trial solution. Initially the value of T is large, giving a worse candidate a fair chance of being selected. The idea is to gradually decrease T thereby decreasing the chance that trial solutions that are much worse will be selected. In this way the heuristic starts with finding a local optimum and are allowed to search for other solutions in the entire solution domain in the beginning, later the search are constrained around the different local optimum hopefully finding the optimal solution before the stop criteria is reached.
Simulated annealing starts with an initial solution. Then moves on to a neighbourhood solution as described above. After a fixed number of iterations T is decreased and the search starts over. This is continued until T has reached a fixed minimum size. Then the best solution found after a fixed number of iterations is selected.
As in the tabu search simulated annealing needs some parameters to be specified before it can be used to solve a specific problem. These are:

- How to select the initial trial solution.
- Define the neighbourhood and how to reach the neighbours.
- How to select a random neighbour to become candidate to the next trial solution.
- The appropriate temperature scheme for T.

1.4.3 Genetic algorithms

A genetic algorithm is somewhat different from the two previous metaheuristics. It is particularly efficient at exploring solutions in the feasible region evolving towards the best feasible solution. The ideas of genetic algorithms belong to the theory of evolution formulated by Charles Darwin. There is a set of individuals, these all have some good and bad genes. When two of these individuals generate a child, this child will inherit the genes from the parents. In some cases it will inherit mostly good genes, but in other cases this may not be the case. When using a 'survival of the fittest' approach, only the children with the best genes will survive and be used for reproducing the next generation. Sometimes mutation of genes takes place. This could result in better or worse genes, with only individuals with better genes being available to evolve during time. In some cases mutation of genes is the reason for some species survival when changes in their surroundings occur. In genetic algorithms mutations are used for creating better solutions. With this scheme the population in general increases in feasibility over time, hopefully generating a super individual that is the optimal solution.
A general iteration for genetic algorithms start with an initial population, then some parents are selected randomly or according to some scheme, and one or more children are constructed, with some mixture of the genes from the parents. Then the entire initial population or some part of this is replaced by the newly created children, and the algorithms start over. This is continued until some sort of stop criteria like the ones mentioned in tabu search is met.
Like in the other two metaheuristics there are some choices that need to be made before applying genetic algorithms to a problem:

- The population size.
- Creating an initial population.
- How to select parents.
- In what way should the parent's genes be mixed to create a child?
- How to replace individuals in the population.
- How to mutate a solution.
- Which stop criteria to use?

Genetic algorithms are the ones used for most of the previous work done in solving the aircraft landing problem.

1.5 The goal of this project

This project consists of two main parts. The first part is to obtain the same results as described in the basic article about the static problem [2]. Here an algorithm needs to be implemented and the results obtained are compared with the original results to see if it is possible to reproduce their experiment. Finally this will be achieved with tuning the algorithm until the results are the same or very similar. The second part is about developing the algorithm created in the first part into a new algorithm that can test different approaches in solving the aircraft landing problem dynamically.

1.6 Roadmap

Section 2 summarizes the previous work on this problem.

Section 3 presents the solution model and describes the experiments.

Section 4 summarizes the result obtained from the experiments.

Section 5 is the conclusion of this project.

Chapter 2

Previous work

This chapter contains descriptions of previous work done in investigating different techniques for solving aircraft landing scheduling and one description of an aircraft departure scheduling. For each article there will be a short summary of the essence of the specific article and the choices made in adapting the different meta-heuristics to the specific problem will be described. There will be a difference in how detailed the different heuristics are described; this is due to the fact that not all of the articles explain their heuristics in detail. At the end of each summery there will be a small evaluation section, where the articles will be evaluated due to their usefulness in consideration to this report.

Notation

Since most of the notations in the articles are the same, these will shortly be described prior to the summary of the different articles.

P = the number of planes.
E_i = the earliest landing time for plane $i (i = 1, \ldots, P)$.
L_i = the latest landing time for plane $i (i = 1, \ldots, P)$.
T_i = the target (preferred) landing time for plane $i (i = 1, \ldots, P)$.
S_{ij} = the required separation time (≥ 0) between plane i landing and plane j landing at the same runway (plane i lands before plane j).$(i = 1, \ldots, P; j = 1, \ldots, P)$.
g_i = the penalty cost per unit of time for landing before the target time T_i for plane i.
h_i = the penalty cost per unit of time for landing after the target time T_i for plane i.
x_i = the actual (scheduled) landing time for plane i.

2.1 Scheduling Aircraft Landings – The Static Case

In this paper [3] dealing with the static case of scheduling aircraft landing their goal is to minimize the total cost of the landings when an aircraft incurs a penalty cost if it does not land at its target landing time. The authors present a heuristic solution and compare that solution with an optimal solution found using a linear programming-based tree search. The authors argue that the model presented by them is so general that it also can be applied to mixed problems with landing and takeoffs, or only takeoffs. This can be done since their model only decides times for planes with respect to time window constraints and separation time constraints. The test conducted has between 10 and 50 aircrafts and 1 to 4 runways. For each test the authors first tried to find an optimal solution using the heuristic, and compared this solution to the actual solution. When the heuristics could not find an optimal solution, the best found heuristic solution was used as an upper bound for tightening the relaxation for the next test, until the optimal solution was found. The algorithm works as follows:

1. Apply the heuristic to generate an upper bound.
2. Use to tighten the time window.
3. Use tree-search to resolve the LP relaxation of the problem:
   (a) If an improved feasible solution is found within a fixed number of seconds then terminate the search and update and restart the search at 2.
   (b) Else continue the tree search until normal termination (an optimal solution found).

Objective function

$$Minimize \sum_{i=1}^{P} (g_i \cdot \alpha_i + h_i \cdot \beta_i)$$

Each test problem is solved with an increasing number of runways until the optimal solution value (the sum of the penalties) is zero and indicating that the algorithm has sufficient number of runways to solve the problem optimal. Unfortunately there are no details about their heuristic other that they use one. This makes it very hard to try to reproduce their experiment even though all their test files and results are available.

Evaluation

This paper has the advantage that all the test files are available on the internet. These files can be used to test the results of an algorithm. Regrettably there are no details about how the authors have implemented their algorithms making it impossible to reproduce their experiments to see if the same results can be obtained.

2.2 Scatter Search and Bionomic Algorithms for the Aircraft Landing Problem

[4] In this paper the static aircraft landing problem is considered. The objective is to achieve effective runway use. Two different heuristics techniques are presented: scattered search and bionomic algorithm. These techniques are subtypes of population heuristics (genetic algorithms) which is described in the introduction.
The tests are completed with two different objective functions. In the first a non-linear objective function is applied and the penalty for landing after T_i is -$(x_i - T_i)^2$. On the other hand there is a positive gain if the aircraft lands before T_i, namely +$(x_i - T_i)^2$. Here the objective becomes to maximize the sum of the function values for all the aircrafts. In the second a linear objective function is applied where there is a positive penalty for landing after T_i and a negative gain for landing before T_i. The overall objective becomes to minimize the sum of the function values for the aircrafts.
The unfitness function is calculated as the sum of the violation of the constraints:

$$\sum_{i=1}^{P}\sum_{j=i}^{P} max(0, S_{ij} - (x_j - x_i)$$

For both the objective functions the test are conducted with up till 500 aircrafts and 5 runways.

2.2.1 Heuristics details

Initial population

Their initial population consists of 100 individuals where three individuals (solutions) are made using E_i , T_i and L_i. The first solution consisting of the earliest possible landing times for each aircraft, the second solution with the target landing times for each aircraft and the third solution with the latest landing times for each aircraft. This is possible for each runway in use since the 3 individuals can be scheduled at each runway separately. The other 100-(3*number of runways) individuals are generated randomly.

Parent selection

In scatter search three different individuals that are to be used as parents are found using binary tournament. Binary tournament selects two randomly individuals and returns the fitter of these. This is done three times for finding the three fittest parents out of three pairs of parents randomly selected. In bionomic algorithm the parents are select in a scheme were the fitter of the individuals have a greater chance of being selected

and individuals that are too similar are prevented for being selected as parents together. In this way at least 100 parent sets are created.

Child generation

In both procedures one child is generated from a parent set. In the non-linear objective function, the new values of the child are a weighted linear combination of the parents, meaning that the better fit of the parents has most influence on the child. In the linear objective function random weights are used in the linear combination process for creating diversity in the new child, here the fitter parent does not have a larger influence on the child. In scatter search only one child is generated at each generation. In bionomic algorithm several children are generated. Hence scatter search calculates the new child faster but need more iterations than bionomic algorithm before the initial population is totally replaced.

Population replacement

For both the procedures the population size is constant and hence one individual needs to be removed when a new is added. In scattered search the one removed when a new child is added is the current worst individual being the one with the highest unfitness value or if there are several with the same highest unfitness value the one of these with the worst fitness value, depending on the objective function. In bionomic algorithm only the best of the generated children are added and the removal is done in the same manner as for the scattered search.

Mutation

For each individual solution the optimal landing time for the scheduled order are made. This is done with a polynomial bounded procedure for the non-linear objective function and as a simple linear problem for the linear objective function.

2.2.2 Results

For the non linear objective, the results indicate that bionomical algorithm outperformed the scattered search. In the linear objective the reverse is true. The largest computation time presented in this paper is the bionomic algorithm used for 500 aircrafts one runway problem. Here it took 1.2 second per aircraft to find the solution indicating that the problem could be solved in 10 minutes.

Evaluation

The algorithms show how different versions on genetic algorithms can be implemented. From the paper it would be possible to conduct an experiment with the same algorithms if the test files were available. This is regretfully not the case.

2.3 A meta-heuristic approach to aircraft departure scheduling at London Heathrow airport

[5] This paper deals with the departure scheduling instead of landing scheduling as the other papers. The aim of the paper is to create a meta-heuristic solution that can be used as a supporting tool for the runway controller. The objective of the solution is to maximize runway throughput. The scope of this paper is only the operations made by the runway controller; the operations of the ground controller and the delivery controller are outside the system.
Aircrafts need to be positioned at holding points before entering the runway. Hence there is a limitation in how the planes can be reordered after they have entered these holding points. Compared to the landing schedule this is a huge restriction since it is much easier to reorder planes in the air than on the ground. This is the fact since in air, aircraft can overtake other aircraft by flying over, under or besides these. At ground the aircraft need to maneuver around other aircraft at a much more limited space.
The separation constraints are somewhat different from the ones in aircraft landing. There is still the separation due to the wake vortex (turbulence) created behind an aircraft, but there is also separations constraints depending on the departure route for each aircraft and separations constraints depending on the speed group of the aircrafts.

Objective function

The objective function is to maximize the runway throughput with all the constraints met. This is obtained by making a model where the delays for each aircraft at the holding point are minimized.

2.3.1 Search algorithms

This paper tests different meta-heuristics, these will be described in the following section.
All of the heuristics have the same basic format but differ in the details. The basic algorithm is as follows;

1. Obtain an initial solution. This is typically a solution with the aircraft in the order they arrived at the holding point.
2. Heuristically assign holding point routes to each aircraft.
3. Check the feasibility of the solution.
4. Evaluate the cost of the solution.

5. Accept or reject the candidate solution. This is where the metaheuristics searches differ. If solution is accepted then it is the new candidate solution.

6. If the given number of evaluations is reached then stop and report the best result so far, else return to step 2.

First descent

This is the simplest algorithm. Each new solution is accepted if it is better than the current solution.

Steeper descent

In this algorithm 50 candidate solutions are selected at a time, then the best candidate solution is selected regardless of this is better or worse than the current solution. In this way additional solutions are selected as the best candidate solution giving the opportunity for other searches when a local optimum is found.

Tabu search

This algorithm is similar to the steeper descent algorithm except that it maintains a list of tabu moves. For each solution the reverse move is added to the tabu list to ensure that the search does not return to a previous investigated solution.

Simulated annealing

Simulated annealing is similar in structure to the fist descent algorithm with the difference that it sometimes accepts moves to worse solutions. A better solution will always be accepted, but a worse solution has a chance of being rejected.

2.3.2 Heuristics details

Three dataset consisting of historical data was used for the incoming aircrafts. The initial schedule was build as follows;

1. Add the first 20 aircrafts.

2. Run the search algorithm for 10000 times, keep the best result found.

3. Fix the first 5 aircraft according to the solution found.

4. Add the next 5 aircraft to the system.

5. Run the algorithm for 5000 evaluations, keep the best result.

Parent selection

Depending on the search algorithm as described above.

Child generation

Again this is dependent on the search algorithm.

Population replacement

After having run the algorithm and found the best solution, the next aircraft is added to the system and the first aircraft in the schedule (solution) is removed. Then the algorithm runs 500 evaluations and the best result is kept. This continues until there are no more aircrafts to add.

Mutation

There are no mutations of the solutions.

2.3.3 Results

The heuristic solution gave much better result than the manual solution this is partly because the runway controller only tries to maximize the throughput and not to minimizing the overall delay. Among the heuristics tabu search outperformed the others but with a small margin.

Evaluation

This paper is useful since it compares different meta-heuristics and is used on another problem than the aircraft landing problem. This gives a better understanding in how meta-heuristics can be applied to different problems.

2.4 Scheduling aircraft landings at London Heathrow using a population heuristic

[2] This paper reports the results of an investigation undertaken by National Air Traffic Service in the UK in improving the runway use at London Heathrow airport. The algorithm used is based on a population heuristic (genetic algorithm) and the algorithm is described in details. Since this paper deals with a specified problem and not just the general problem there are many details in the algorithm, some of the most useful for understanding the process are the table of the separation constraint between aircrafts of different type and the standard landing speed for all aircrafts.
The algorithm was tested on different data sets related to landings at London Heathrow. Only the result of one of these sets was shown in the paper. This set contained 20 aircrafts and they were all within 100 nautical miles of Heathrow. This particular set was chosen since it was considered a special busy period.

Objective function

The objective function tries to maximize the total fitness value:

$$\sum_{i=1}^{P} Z_i \text{ where } Z_i = \begin{cases} -(D_i)^2 if D_i \geq 0 \\ +(D_i)^2 \text{ otherwise} \end{cases}$$

$$D_i = (x_i - T_i$$

This implies that a solution where the aircraft lands before there target time is preferred.
The unfitness is calculated as:

$$\sum_{i=1}^{P} \sum_{j=1}^{P} \max \left[0, S_{ij} - (x_j - x_i)\right]$$

When a solution is feasible, meaning that all constraints are met, then the unfitness values equals zero.

2.4.1 Heuristic details

Initial population

The initial population size was 100, with all solution being randomly generated.

Parent selection

Randomly select two individuals, the one individual with the best fitness value, is selected as a parent. Then repeat this selection again to find the other parent.

Child selection

For creating a child, uniform crossover was used on the two selected parents.

Population replacement

A steady state population scheme was adapted, for each child created one individual was removed from the population. This was done by ordering all the individuals according to the new child into four different sets.

1. One set called G1 contained the individuals that had worse fitness and worse unfitness value.
2. The second set called G2 contains the individuals that are better in fitness but worse in unfitness values.
3. The third set called G3 contains the individual with worse fitness and better unfitness values.
4. The fourth set called G4 contains the individuals with better fitness and better unfitness values.

Then the algorithm finds a random individual in the first set containing individuals, starting to look in G1, then G2, then G3 and finally G4.

Mutation

It was experimented, but did not give any significant gain in result.

2.4.2 Results

The overall results of the tests showed that the result could be improved by 2-5 percent. The average delay was reduced by 41 seconds, but the maximum delay was increased by 187 seconds.

Evaluation

This paper is the most interesting for the project since the authors describes in details how their population heuristic is implemented. The problem with this article is that the test files are not available, making it impossible to copy the project and getting the same result.

2.5 Displacement problem and dynamically scheduling aircraft landings

[6] This paper handles the dynamic problem with new aircraft appearing dynamically. The original static solution needs to be resolved with some constraints regarding the amount of changes. This is handled by a displacement function that calculates the penalty for making changes.

2.5.1 Solving the displacement problem

The authors use the same algorithms as they did in the "Scheduling Aircraft Landing - the static case" article.
By making suitable changes to the algorithms of the static aircraft landing problem these can be used for solving the dynamical problem. Linear problem (LP) based tree-search as presented in the other article can be directly applied in order to find the optimal solution to each displacement problem. The population heuristic algorithm needs some adjustments. Therefore the authors:

- Extend the representation together with the crossover and mutation operators, to include runway choice.
- Incorporate maximum displacement by time window modification.
- Seed the initial population with suitable individuals based upon the landing sequence given by sorting aircrafts into earliest/target/latest time order.
- Apply the population heuristic after first using the original LP based tree-search and seeding the initial population with the best solution found by the LP based tree-search.
- Take the best solution as found by the population heuristic and solve the aircraft landing problem to decide landing times.

Objective

The displacement function $D_i(\underline{X}, \underline{x})$ is solved by summing all the changes made to the original schedule $\underline{X}$ by the new schedule $\underline{x}$. The value of the change is based on whether the original scheduled landing time was before, the same or after the target landing time.

$$\begin{aligned} D_i(\underline{X}, \underline{x}) &= g_i * max(0, X_i - x_i) if X_i < T_i \\ &= h_i * max(0, x_i - X_i) if X_i > T_i \\ &= g_i * max(0, X_i - x_i) + h_i * max(0, x_i - X_i) if X_i = T_i \end{aligned}$$

The objective is to minimize $D_i(\underline{X}, \underline{x})$

2.5.2 Results

For smaller problems the LP tree-search is better than the heuristic. There exists some cases were the population heuristic gives a better solution; this is due to the sequence of the next incoming aircrafts. In problems with a large number of aircraft the population heuristic solution is superior, and often the LP approach could not find a solution (timeout).

Evaluation

This is the only article that deals with the dynamic problem of scheduling aircraft landings. This gives insight in one way that the dynamic problem can be solved. The results are based on files available on the internet making it possible to compare their result with average delays and maximum delay of the rescheduled solution.

Chapter 3

Model

3.1 Remaking of previous experiment

In this section one of the experiments described in the chapter of previous work is implemented to test if the same results are possible to achieve. Hereby validating the implemented model. First there is an argumentation for why that specific experiment is selected followed by a detailed description on how to implement the chosen meta-heuristic. Finally the results obtained from the tests conducted, is compared to the test result available on the internet.

3.1.1 Choice of experiment

The experiment remade steams from [2]. The selection of the experiment is due to that the test files available on the internet giving the possibility for comparing the result, with the ones obtained in the remake of the experiment. Since there are no details about the heuristic algorithm used by the authors in the experiment mentioned above, the heuristic from the experiment [3] is used. The reason for this choice is the detailed description on how this heuristic was implemented.
This chapter is divided into two parts. The first part is about implementing the heuristic and testing this on some simple test files that are constructed for this case. These tests are conducted to find suitable parameters for the population heuristic regarding:

- population size
- number of iteration without creating better solutions
- number of restarts of the algorithm for creating diversity in the results

In the second part of the chapter the heuristic is tested on the test files from [2] available at the OR-library on the internet [7]. First the heuristic

is tested directly on the files to see if the results are close to the ones from the original experiment. Secondly, the implemented heuristic is used as the heuristic in the algorithm from the original experiment where it is used for tightening the problem and thereby finding better solutions.

3.1.2 Implementation

The algorithm can be implemented in several ways. Obviously the choice of selected language for programming plays a major role in the methods for implementation of a given algorithm. Generally the implementation can be done in two different ways either using a mathematic language such as Matlab, Maple or Mathematica. Another approach is to use a standard programming language such as C, C++, C#, Java etc. The choice is on an object oriented programming language, all of the details presented later in this section can be used in any object oriented language of choice. The implementation for this project is in Java. The reason for this choice is my experience with this language.

Outline for the heuristic algorithm

The goal of the heuristic is to maximize the fitness value and generate a feasible solution (a solution where the unfitness value is zero).

Listing 3.1: the heuristic

```
j=0, best.fitness = 0.
Create initial population

while (j < max_iterations)
        Find parent solutions
        Create child solution
        Replace a solution in the population with the child
        if (child.unfitness==0 & child.fitness > best.fitness)
                best = child
                j = 0
        else
                j = j+1
        end
end

return best
```

Representing a solution

The class named solution consists of one schedule for the aircraft created for a specific problem. A solution hence consists of a list of aircrafts and the variables fitness value and unfitness value.

Representation of an aircraft

This is done in a class named aircraft. The variables are earliest landing time, target landing time, latest landing time, scheduled landing time and what type of aircraft the class represents.

Fitness function

In an evaluation class the fitness value of a solution is calculated. The fitness function is calculated to maximizing the runway throughput. For this specific problem a non linear objective function is made as a method. The non linear objective function is calculated as

$$\sum_{i=1}^{P} Z_i \text{ where } Z_i = \begin{cases} -(D_i)^2 if D_i \geq 0 \\ +(D_i)^2 \text{ otherwise} \end{cases}$$

$$D_i = (x_i - T_i$$

Unfitness function

Like the fitness function the unfitness function is calculates in a method in the evaluation class. The unfitness is calculated as;

$$\sum_{i=1}^{P} \sum_{j=1}^{P} \max \left[0, S_{ij} - (x_j - x_i)\right]$$

Whenever the constraint is fulfilled the value added to the sum is zero. If all the constraints are fulfilled the total sum is zero making it a feasible solution.

Creating an initial population

The initial population consists of a number of randomly generated solutions. For each new solution the scheduled landing time for each aircraft is made using a random function that assigns the value of the scheduled landing time to a value in the time window for that aircraft. This can typically be done like this; scheduled landing time = earliest landing time + random(time window). Creating an initial population is done in the *createInitialPopulation* class whit the size of the initial population given as a parameter.

Child generation

For creating a child two sets of parents need to be selected. The selection scheme is done by binary tournament where two candidates first are selected at random. The one of these with the best fitness value is selected as the first parent then two additional candidates are selected at random and the one of these with the best fitness value is selected as the second parent. After having found the two parents a child is generated by randomly selecting the scheduled landing time from one of the parents for

each aircraft in the solution. The child generation is done in a class called *populationHeuristic*.

Replacement scheme

This is where most of the calculation takes place. For each newly generated child, the currently 100 solutions in the population need to be ordered accordantly to the child. This is done by first making four empty sets named G1, G2, G3 and G4. Then each solution in the population is placed in one of the sets according to this scheme: if fitness value and unfitness value both are worse than the values for the child the solution is placed in G1. In the case were the unfitness value is worse and the fitness value is better then the solution is placed in G2. If the fitness value is worse but the unfitness value is better then the solution is placed in G3. Finally if both fitness and unfitness values are better then the solution is placed in G4. After this ordering a solution randomly selected from G1 is removed, if G1 is empty then a randomly selected solution from G2 is selected. Similarly if G2 is empty a solution from G3 is selected. If there are no solutions in G1, G2 or G3 then a solution in G4 is selected to force a change in the population.

Changes made to the algorithm for the second part of the experiment

First the heuristic is changed a little since the objective function changes and the goal now is to minimize the fitness value. The representation of an aircraft is changed so that the penalty for landing earlier or later than the target landing time can differ and the way the separation are handled are somewhat different. For the following tests a new class named *AircraftExt* is used instead of the original *Aircraft* class.

New fitness function: minimize

$$\sum_{i=1}^{P}(g_i * \alpha_i + h_i * \beta_i)$$

Listing 3.2: new heuristic

```
j=0, best.fitness = 0.
Create initial population

while (j < max_iterations)
        Find parent solutions
        Create child solution
        Replace population with child
        if (child is feasible & child.fitness < best.fitness)
                best = child
                j = 0
        else
                j = j+1
        end
end

return best
```

If the result of this algorithm does not perform satisfactory another approach will be tested namely:

1. Apply the heuristic to generate an upper bound Z_{UP}
2. Use Z_{UP} to tighten the time window.
3. Finding stop criteria
   (a) If an improved feasible solution is found within a fixed number of seconds then terminate the search and update Z_{UP} and restart the search at 2.
   (b) Else return the best found solution.

3.1.3 Test

Here different tests for the static problem will be conducted. The first tests are simple to investigate if the algorithm can get results that are useable compared to an optimal solution. Here the algorithm will be tuned for performance by testing different stop criteria. The third test will be the same as described in the "Scheduling Aircraft Landings - The Static Case" article. This is done so that the algorithm can be tested according to a large set of different tests with the possibility to conduct a comparison with the results obtained by the authors of that article.

Test1

In this test a very small problem will be solved with only five aircraft. The time window for the different aircraft overlap in some extent forcing the algorithm to make some replacement in the population before a feasible solution is found. This test is conducted to see if the algorithm can create a feasible solution. This implies that the goal of this test is to create a solution with unfitness value of 0. The results can be seen in figure 3.1

Test2

This test contains 10 aircraft. The distance between their earliest landing times are the minimum separation constraints between each of the aircraft. The goal of this test is to have a reference test used for fine tuning the algorithm.

Tuning the algorithm

For tuning the algorithm there are three main variables that can be changed, these are;

- Number of iterations without improvement: Δ
- Size of the initial solution: Θ
- Number of restarts, giving different solutions: R

The first variable is an indication of how many children there will be created without creating a child that is better than the best solution found so far. The second variable indicates how large the initial population is.The larger size, the greater diversity there are in the solutions. The third variable determents the number of times that the test should be restarted. This variable should be large enough to ensure that the algorithm at least finds one, and preferably several feasible solution from where the best can be selected.

Deciding on values

Based on the results from test2 (see table 3.2 - table 3.4), the following values for the main variables are decided;

- $\Delta = 5000$
- $\Theta = 100$
- R = 10

Test3

Here some of the files from [2] are used as a reference and the result are compared with the results from the article. For this testing both the objective function and the replacement scheme needs to be changed, since the goal is to minimize the objective instead of maximizing it as done in the previous tests.

Tuning the algorithm

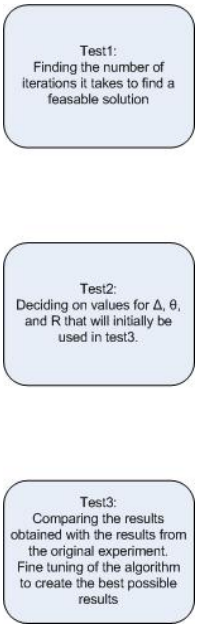
In the first test, the results were not near the optimal solution presented in the authors' article namely a solution with fitness value of 700 with only one runway available. Even after several attempts of tuning the algorithm, the calculated results were still 50 times larger than the optimal value. This indicated clearly that it was not enough just to use the selected algorithm directly on a different problem. The best result with the same choices of the main variables from test 2 was: 36650. (See table 3.5)
Now the obvious approach was to use the heuristic in the same way as the authors did in their experiment. Namely finding the best possible solution in a given amount of iteration, and solve the problem again with this solution as an upper bound for the next solution and hence tightening the problem. In practice this was done by changing the Elt or the Llt to the scheduled time found in the best solution so far. If the found solution had a scheduled time before Tlt, then the Elt was changes to the scheduled value otherwise it was the Llt. Then the heuristic was applied again and a new best solution was used to tightening the problem. This continues until some stop criteria was reached.
As the results indicates (see table 3.6 - table 3.11), the latter approach did increase the result dramatically. Still the optimal solution values were not found after having tested with different values for θ, δ and R.

By mutating the final result these could be improved some but still not lead to the optimal solutions. (see table 3.12)

It is hard to point out why the obtained solutions are somewhat different from the optimal solutions. One reason could be that the authors of the original experiment used another replacement scheme. Another reason could be that they mutated their solutions in a way that was more useful and the mutation used in this remaking of their experiment. The most obvious reason could be that they used another metaheuristic than a population heuristic and this heuristic proved more useful for their experiment.

3.1.4 Results

The tests conducted are the ones decribed in the privious section. The main objective for each of these test are seen in figure 3.1.4.



Test1

The results shows how many iterations it took for the algorithm to find a feasable solution. The same test is conducted 5 times and the results are listed in table 3.1.

Table 3.1: Test1

nr:	1	2	3	4	5
Iterations before finding a feasible solution:	305	179	319	154	20

Test2

First the value of the optimal solution was calculated manually to 253857. This was done to have a optimal solution for the heuristic to compared with. The following tests were all conducted with an initial population of 100 solutions. This value was chosen since the authors of [3] in their algorithm uses this value for their population.
Since there are some diversions in the quality of the found solution for each test run, it seems that restarting the algorithm several times would give an

Table 3.2: Test2.1 $\theta = 100$

nr:	$\Delta = 200$	$\Delta = 500$	$\Delta = 1000$	$\Delta = 2000$	$\Delta = 5000$	$\Delta = 10^3$
1	164505	3450	118197	207990	246870	235896
2	88310	95500	213054	216950	232700	242766
3	41901	119242	219739	225190	229457	236530
4	nf	164309	175697	241073	252804	219738
5	nf	65176	194510	229176	243884	242294
6	nf	212938	218706	234489	240434	245318
7	nf	189247	164730	244474	236721	249898
8	nf	142989	201737	236535	235788	236121
9	143354	55547	174773	228946	245577	243155
10	146955	75449	142086	250195	248126	240725

useable best solution. The choice is to use 10 restarts. This should guarantee that at least one feasible solution is found and that the best found solution is acceptable. As the results shows (see table 3.2) choosing a value for δ to 5000 or 10000 seems useable. This is then tested again with 10 restarts of the algorithm for each test and the best result is compared with the optimal result (see table 3.3).

Table 3.3: Test2.2 R = 10

nr:	$\delta = 5000$	% of optimal solution	$\delta = 10000$	% of optimal solution
1	237892	93.71	208328	82.07
2	253614	99.90	211816	83.44
3	243482	95.91	244590	96.34
4	234225	92.26	250786	98.79
5	244914	96.48	250366	98.62
6	226960	89.40	253103	99.70
7	247982	97.69	231986	91.38
8	250381	98.63	242415	95.49
9	234195	92.25	232856	91.73
10	253158	99.72	248522	97.90

Since there is no significant increase in the result when having 10000 instead of 5000 iterations before the stop criteria is found, $\delta = 5000$ is selected.

Finally different value for the initial population θ was tested. For these tests $\delta = 5000$ and the number of restarts is 10 (see table 3.4).

Table 3.4: Test2.3 different values for θ

Init pop	$\theta = 20$	$\theta = 50$	$\theta = 80$	$\theta = 120$	$\theta = 150$	$\theta = 200$
Calc Best	nf	197546*	173964	185822	217304	203597
Init pop	$\theta = 500$	$\theta = 1000$	$\theta = 2000$	$\theta = 3000$	$\theta = 4000$	$\theta = 5000$
Calc Best	208594	250233	252958	247766	253393	246918

* It was not possible to obtain a feasible solution with $\theta = 20$, while the result where $\theta = 50$ had to be calculated 5 times before a feasible solution were found.

The results indicate that a larger initial population gives better results. Unfortunately the computational time increases dramatically for larger values of θ. Since the aim of this project is to develop a solution for the dynamic problem, computational time is of big importance, hence the value of θ is selected to 100.

Test3

First the algorithm from test2 was used directly only changing the objective function (see table 3.5). The test is conducted on three different dataset. The tables are constructed with the following information:

- test run: the number of the test.
- Dataset, optimal solution / best heuristic solution found in [2].
- nf : solution not found or not feasable.

Table 3.5: Test3.1 using the algorithm from test2

Test run	Dataset 1: 700 / 700	Dataset 2: 1480 / 1500	Dataset 3: 820 / 1380
1	40680	nf	nf
2	44060	nf	nf
3	42480	nf	nf
4	46620	65190	nf
5	47460	71180	nf
6	46670	nf	nf
7	42810	nf	nf
8	39090	72500	nf
9	42290	nf	nf
10	43650	nf	nf

Secondly the algorithm was changed to use the previous solution as the offset of a new solution and thereby tightening the problem.

Table 3.6: Test3.2 tightening the problem

Test run	Dataset 1: 700 / 700	Dataset 2: 1480 / 1500	Dataset 3: 820 / 1380
1	2200	32270	nf
2	2460	nf	nf
3	2740	nf	nf
4	2390	6710	nf
5	1630	nf	nf
6	3470	nf	nf
7	2170	nf	nf
8	3260	68650	nf
9	1920	nf	nf
10	2400	nf	nf

The results for the first dataset improved dramatically. For the other two datasets there are still problems with finding a feasible solution (see table 3.6).

After having confirmed that a significant better solution can be found by tightening the problem, the main variables are again tested with different values to see if another increase in the results can be found (see table 3.7 - 3.11).

Table 3.7: Test3.3 $\delta = 5000$, $\theta = 500$ and R = 10

nr	Dataset 1: 700 / 700	Dataset 2: 1480 / 1500	Dataset 3: 820 / 1380
1	1600	7340	nf
2	3270	nf	nf
3	1880	nf	nf
4	2090	nf	nf
5	1480	nf	nf

Table 3.8: Test3.4 $\delta = 5000$, $\theta = 1000$ and R = 10

Test run	Dataset 1: 700 / 700	Dataset 2: 1480 / 1500	Dataset 3: 820 / 1380
1	3620	nf	nf
2	1420	nf	nf
3	1610	nf	nf
4	5260	nf	nf
5	1540	nf	nf

Table 3.9: Test3.5 $\delta = 10000$, $\theta = 100$ and R = 10

Test run	Dataset 1: 700 / 700	Dataset 2: 1480 / 1500	Dataset 3: 820 / 1380
1	2520	nf	nf
2	2520	nf	nf
3	2520	5590	nf
4	1730	5280	nf
5	2130	nf	nf

Table 3.10: Test3.6 $\delta = 10000$, $\theta = 500$ and R = 10

Test run	Dataset 1: 700 / 700	Dataset 2: 1480 / 1500	Dataset 3: 820 / 1380
1	2350	6090	nf
2	1640	5140	3080
3	2140	2520	5670
4	1820	3010	nf
5	1700	3760	6610

Table 3.11: Test3.7 $\delta = 10000$, $\theta = 1000$ and R = 10

Test run	Dataset 1: 700 / 700	Dataset 2: 1480 / 1500	Dataset 3: 820 / 1380
1	4030	4390	nf
2	1350	4800	nf
3	1320	nf	nf
4	1950	nf	nf
5	1890	nf	nf

Based on the results the main variables are set to $\delta = 10000$, $\theta = 500$ and R = 10.

Finally the final found solution was mutated in a simple manner by using any excess space between two aircraft when considering the separation constraint and then change the scheduled landing time towards the target landing time. Each result first shows the original found solution and then the mutated solution (see table 3.12). The results in row 2 of table 3.12 are read: original value / mutated value.

Table 3.12: Test3.8 simple mutation

Test run	Dataset 1: 700 / 700	Dataset 2: 1480 / 1500	Dataset 3: 820 / 1380
1	1420 / 1420	5890 / 5250	nf
2	1870 / 1870	5350 / 4610	nf
3	1940 / 1940	3060 / 2880	nf
4	1990 / 1900	5170 / 4680	10330 / 3210
5	5170 / 4690	3700 / 3450	7410 / 4930

The java files for test 1-3 are placed in appedix A.2. The .txt files are placed in appendix A.1

3.1.5 Conclusion

In the remaking of the privious experiment it was not possible to get usable results when comparing with the first developed algorithm. After having created another algorithm where the heuristic was used for tightening the problem until this no longer was possible the results were in the same area as the original results but it was still not possible to find the optimal solutions. In some of the test files the used algorithm had problems finding a feasable solution if the initial population size was to small or the number of iterations without improvement was not large enough. Based on these results it can be concluded that the heuristic used were not optimal. There are no description on what type of heuristcs the authors of [2] had used therefore the population heuristic created may not be able to get better results, this may be due to the population replacement scheme used. Another metaheuristic might have generatede better results, this was not implemented since the main reason for developing a metaheuristic was to have a solution creating tool to use in the dynamic case.

3.2 Dynamic scheduling of the aircraft landing problem

As mentioned in the introduction the main difference between the online / dynamic problem and the offline / static problem is the information available before scheduling. In the static case dealt with in the previous section all the information about each of the planes are available and an optimal solution can be calculated before any aircraft are airborne. In the dynamic case there is no guarantee that the planes will arrive at the expected time. This could be due to weather conditions, problems with the crew being late or other delays making it impossible for the aircraft to arrive at the airport at the expected time. The most common reason for an aircraft not arriving at its scheduled time is due to delays. In some cases the aircraft appears before the expected time this could be due to optimal weather conditions.
Before starting on solving the dynamic case a preliminary schedule need to be defined. This schedule can be calculated as a static case. The dynamic case thus becomes a problem on how to reschedule the aircraft when one or more aircraft cannot arrive at the scheduled time. Since it is common that the aircraft do not arrive precisely at the scheduled time the rescheduling will be calculated numerous times. Therefore it is not practical to reschedule the entire schedule when one aircraft misses its scheduled landing time. An approach on how to solve this could be to only reschedule a part of the original schedule. The aim of this project is to analyze different ways on deciding what part of the schedule to reschedule when aircrafts are late or early, and compare the different results. The problem can be seen as a displacement problem where there are penalties for moving the aircrafts from their original schedule and the objective is to minimize this total penalty.

The following constraints must be fulfilled:

- The new schedule must be a feasible solution.
- A rescheduled aircraft other than the one causing the reschedule cannot land before its original scheduled landing time.
- Each aircraft need a landing time before it's latest landing time. The latest landing time is in this project decided to be 30 minutes after the aircraft's scheduled landing time.

When the constraints are fulfilled there still are some parameters that will contribute to the result of the reschedule.

These are:

- The number of aircrafts that are rescheduled
- Which aircrafts to reschedule
- Guaranteeing that an aircraft is not rescheduled if it is scheduled to land within a certain amount of time.

Based on these considerations a number of different approaches are possible for dealing with this problem. For all of the possible solution described later in this chapter, a delayed aircraft is removed from the schedule when it is not ready for landing at the specified time. When this aircraft arrive its new landing time will be scheduled within a sequence of the original schedule. It is this sequence of aircrafts that need to be found for each approach, before making the reschedule.
The approaches for how to deal with delayed aircraft are:

1. Letting the number of aircraft being rescheduled be decided by the aircrafts needed for making a specific gap. This means a specified number of free-time (seperation time larger than the separation constraint) that is available in the original schedule before adding the delayed aircraft into the calculation of a reschedule.
2. Having a fixed number of aircraft that are rescheduled when a delayed aircraft appears.
3. Having a fixed period of time, and rescheduling all aircraft that are scheduled to land within this period.
4. Iteratively finding a minimum of aircraft that are to be rescheduled. First try to find a feasible solution with only one aircraft being rescheduled. If this fails try rescheduling with two aircraft etc. This continues until a feasible solution is found.

The different approaches for dealing with aircraft appearing earlier than scheduled are:

I Let the early aircraft circle above the airport until it can land at the scheduled landing time.

II If there is a gap in the schedule were the aircraft can land before its scheduled landing time land there. Otherwise land at scheduled time.

III Remove the aircraft from the schedule and deal with the aircraft as described in the description of the delayed aircraft.

For each of these approaches several tests will be conducted and then the results will be compared to each other to see which combination that gives the best result. The result will be compared regarding to average delay and maximum delay for each data set in the test.

Objective function

For each of the different approaches the objective function is the same as the one used in the second part of the static experiment namely:

$$\text{Minimize } \sum_{i=1}^{P}(g_i * \alpha_i + h_i * \beta_i)$$

This objective function has a penalty α_i when aircraft i is landing before its target landing time and a penalty β_i for landing after its target landing time. g_i and h_i are the number of time units that aircraft i land before or after the target landing time. Since the target lading time equals the scheduled target time and this also is the earliest landing time according to the constraints mention above the objective function now can be written:

$$\text{Minimize } \sum_{i=1}^{P}(h_i * \beta_i)$$

This can be done since one of the constraints for the dynamic case is that an aircraft cannot land earlier than scheduled, therefore the first part of the original objective function will always equal zero and can be removed. The penalty for landing after the scheduled landing time is selected to be the same as the penalty for landing after the target landing time in the static case. This was selected to have a reference for calculating the fitness value solution instead of creating new values for the calculation.

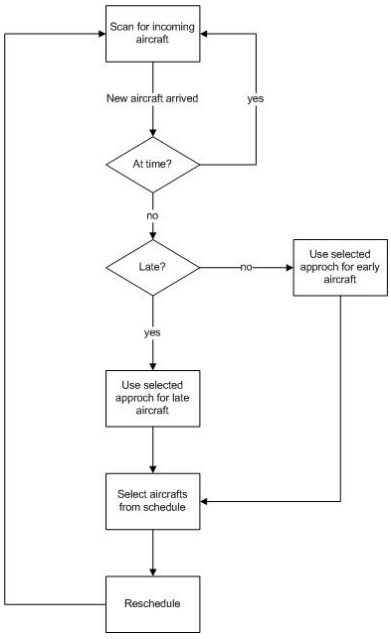
Appearance time

The appearance time could be when the aircraft is within the range of the airport control tower and is asking permission to land. This request is made when the aircraft is within 20 kilometers of the airport. With a landing speed of 160 nautical miles per hour this means that the request should be made 304 seconds or approximately 5 minutes before scheduled landing. For the test made in this chapter the apperance time is decided according to the test files, with the value giving the lowest possible number of late or early aircraft.

Scanning for incoming aircraft

There are several possible choices for selecting how often the program should scan for incoming aircraft. This could be every second, every minute or another fixed interval of time according to how much computational effort the system should use on scanning and how early the reschedule can be calculated. For the tests in this chapter the scan will be performed every time unit.

Outline of the general algorithm



3.2.1 Detailed description of the various approaches

For each of the mentioned approaches from the previous section a more detailed description will be presented here. For each approach there will be a short motivation of why this could be a good solution and then a more detailed description on how it could be implemented will be presented.

1: Finding a gap

Motivation

Minimizing the number of aircraft being rescheduled by only rescheduling the number of aircraft that are needed to find a new feasible solution. Since the maximum separation constraint between two aircrafts are known the gap can be defined as a value times this maximum separation value. A larger gap gives a relaxed problem for the heuristic to solve but requires more aircraft to be rescheduled. Consequently a smaller gap gives a tighter problem and hence can give problems in finding a feasible solution but requires less aircraft to be rescheduled.

Description

By searching the schedule from the time when a delayed or early aircraft appears and continue this search until the added sum of the excess time between two scheduled landings when the separation constraints are fulfilled is large enough (see figure 3.3 for details).

Listing 3.3: outline of approach 1

```
gap=0
i = 0

while (gap < max*alpha)
gap = gap + (aircraft(i+1).scheduledLandingTime -
        aircraft(i).scheduledLandingTime)-
        seperationConstraint(i,i+1)
        i = i+1
end

reschedule (aircraft(0) untill aircraft(i))

Parameters
-       max: Maximum separation constraint
-       alpha: Multiplication_factor
```

2: Fixed number of aircraft for rescheduling

Motivation

Having a fixed number of aircraft that should be rescheduled makes the selection of which aircraft that should be rescheduled very simple. This could give an increase in the calculation time since the algorithm does not

need to calculate which aircraft to reschedule. The number of aircraft that should be selected for rescheduling need to be large enough to give the heuristic the possibility to find a feasible solution but not so large that the heuristic need to reschedule to many aircrafts.

Description

This is the most simple of the different approaches. The selection of aircraft for rescheduling is static and therefore it is very simple to take these out of the schedule and then make the reschedule with the delayed aircraft added to the list for rescheduling. The main problem for this approach will be to find the lowest number of aircraft that are rescheduled still being able to find a feasible solution (figure 3.4).

Listing 3.4: outline of approach 2

```
reschedule (aircraft(0) until aircraft(i))

Parameters
-       i: Number of aircraft for rescheduling
```

3: Fixed period of time

Motivation

The idea is similar to the one presented in the fixed number of aircraft approach. The difference is, that in this approach all the aircraft scheduled to land within a fixed number of time will be selected for rescheduling together with the delayed aircraft just arrived ready to land. This approach is not expected to give good results since the number of aircraft being rescheduled can vary greatly, but the results can be used for comparing this approach with the others.

Description

When a delayed or early aircraft arrives, the schedule is scanned for which aircraft that are scheduled to land within a specified interval after the current time. Each aircraft scheduled to land in this specified time interval is selected for rescheduling together with the delayed aircraft (figure 3.5).

Listing 3.5: outline of approach 3

```
i = 0

while (aircraft(i).getScheduledLandingTime < current_time + t)
        i = i+1
end

reschedule (aircraft(0) until aircraft(i))

Parameters
-       t: Time_interval
```

4: Iteratively finding the number of aircraft for rescheduling

Motivation

Minimizing the number of aircraft being rescheduled, by finding the lowest possible number of aircraft needed for calculating a feasible solution. This approach is somewhat similar to the approach 2 with a fixed number of aircraft being rescheduled. In this approach a reschedule will be made with an increasing number of aircraft until a feasible solution is found. This approach then finds the lowest number of aircraft for the reschedule that gives a feasible solution but need to make the reschedule a number of times. For the tests the number of aircraft rescheduled are simply incremented until a feasable solution is found. Finding the minimum number of aircraft needed for creating a feasable solution can later be changed to a more efficient search if this approach are to be implemented into another system.

Description

When a delayed aircraft arrives, this approach tries to make a reschedule with only one aircraft from the original schedule. If the heuristic cannot find a feasible solution then make the reschedule with two aircraft from the schedule etc (figure 3.6).

Listing 3.6: outline of approach 4

```
i = 0

while (solution.unfitness != 0)
        resechedule(aircraft(0) until aircraft(i)
        i = i+1
end

Parameters
-       i: number of aircraft being rescheduled
```

I: Ignoring an early aircraft

Motivation

Do not reschedule if an aircraft is early. In this way no planes will be delayed if an aircraft arrives early.

Description

If an aircraft is early it must circle around the airport until its scheduled landing time then it may land.

II: Finding a gap where the early aircraft can land

Motivation

This approach gives the possibility for an early aircraft to land before its scheduled landing time if there is a gap in the schedule were it can land and still fulfill the separations constraint without rescheduling any of the other aircraft in the schedule. The underlying idea for this approach is that an aircraft can land before the scheduled landing time if this does not affect the other aircraft scheduled for landing. Otherwise it must wait to land until its scheduled landing time.

Description

When an aircraft arrives earlier than scheduled the schedule is scanned to see it there is a gap between the current time and the aircraft's scheduled time where the aircraft can land. If there is a gap the aircraft is scheduled to land in this gap. Otherwise it must wait until its scheduled landing time (see figure 3.7).

Listing 3.7: outline of approach II

```
i= 0
early = incoming early aircraft

while (i < original number in schedule)
        one = aircraft(i).scheduledLaningTime
        two = aircraft(i+1).scheduledLandingTime
        window = two - one
        sepOne = seperationConstraint(aircraft(i), early)
        sepTwo = seperationConstraint(early, aircraft(i+1) )

        if (window>(sepOne+sepTwo )
                early.scheduledLandingTime = one + sepOne
        end
end

Parameters
-       There are no parameters that can be changed in this approach
```

III: Early aircraft treated as a late aircraft

Motivation

Instead of treating an early aircraft as a special case it is treated as a late aircraft and therefore it is removed from the original schedule and added into the reschedule. The main advantage with this approach is that all aircraft are treated in the same matter. The disadvantage is that some aircraft are forced to land later than scheduled due to an aircraft that arrived early.

Description

The outline will be the same as the selected approach amongst the different approached for dealing with a late aircraft.

3.2.2 Testing

This section will describe the tests that are performed. First there will be a description about how the tests are constructed, then how they are used. This section will give an explanation on how the results are found in the next chapter.
For each of the tests presented here, no aircraft will be rescheduled after its apperance time. Different values for the freeze time that is added to the apperance time will be tested later.
Each approach will be tested on some of the test files available at the OR-library [7] and used by the authors in [2] & [6]. For finding the best approach, the same three test files are used as in the previous chapter. The best possible solutions found in section 4.1.4 for each of the test files are used as the original schedules where the reschedule will depend on the appearance time of the aircraft.

Approach 1

Parameters:

- max = 15 since this is the largest separation constraint in the test files.
- alpha = 0.2, 0.4, 0.6, 0.8, 1.0, 1.2, 1.4, 1.6, 1.8, 2.0

The minimum separation constraints in the test files is 3. This value equals $0.2 \cdot max$, therefore 0.2 is selected as the smallest value for alpha. For values of alpha smaller than 1.0 it is not realistic that the results will be feasible but for some schedules this could give a feasible solution.

Approach 2

Parameters:

- i: 3, 4, 5, 6, 7, 8, 9, 10

The different values of i are selected to test if a small number of rescheduled aircraft can generate a feasible solution. The larger number of i is selected to see if a value for i can be found were the approach always find a feasible solution.

Approach 3

Parameters:

- t: 20, 30, 40, 50, 60, 80, 100

The reason for the selected values is that 15 is the maximum separation constraint and the time period t should be larger than this if this approach should be able to create feasible solutions. The largest value selected for t in this test is sufficient large that even with maximum separation constraints between the aircraft there is the possibility that at least six aircraft can be rescheduled if they arrive with the seperation constraint being the difference in their landing time.

Approach 4

Parameters:

- i: 1, 2...,n

The values of i is here decided iteratively and for each test the maximum value of i will be presented in the results from the test.

Approach I

This approach has no parameters but the results will be compared to approach II & III.

Approach II

Based on the results from approach 1-4 one of these will be selected as the reference approach when comparing the results with the results from approach I & III.

Approach III

This approach has no parameters but the results will be compared to approach I & II.

Freezing time

In this test several values for a freeze time are tested where the aircraft are guaranteed not to be rescheduled if they are about to land within this freeze time. One of the approaches tested above will be used for reference testing with the different freeze time values.
Parameters:

- freeze: 3, 5, 10

The values selected are not dependent on any calculation but are selected to have some different values to use. Still the latest landing time for an aircraft cannot be larger than scheduled landing time + 30 minutes.

3.2.3 Conclusion

For the dynamic test the results for the different early approches showed that dealing with an early arriving aircraft in the same way as if it was late gives better results when comparing the avarage delay. The maximum delay was not influence when dealing with early aircraft as if they vere late, this is probarbly because of the way the aircraft arrive in the testfiles and was somewhat unexpected. For the late approaches 1-4 there are no difference in their results. The reason for this could be that the time window for each of the aircraft are so tigtened when rescheduling that the heursitic always find the same solution. These results were not expected and it is hard to conclude which approach that is better than the others since they all gave the same results. Approach 1 was the one that were expected to give the best results since it would not start the reschedule before there was enough space between the other incoming aircraft.

Chapter 4

Results

4.1 First come first served

This test is made to have reference results for the other tests.
Testfile = airland1.txt

- Average delay = 1.50
- Maximum delay = 2

Testfile = airland2.txt

- Average delay = -14
- Maximum delay = 3

Testfile = airland3.txt

- Average delay = -21
- Maximum delay = 10

4.2 Approach 1

For each test in this section the results are listed in a table with the different values for alpha listed in the upper row, the calculated values for mean delay in row 2 and the maximum delay presented in row 3.

4.2.1 Late approach 1, early approach I

Testfile = airland1.txt

alpha	0.2	0.4	0.6	0.8	1.0	1.2	1.4	1.6	1.8	2.0
Mean delay	1.50	1.50	1.50	1.50	1.50	1.50	1.50	1.50	1.50	1.50
Maximum delay	2	2	2	2	2	2	2	2	2	2

Testfile = airland2.txt

alpha	0.2	0.4	0.6	0.8	1.0	1.2	1.4	1.6	1.8	2.0
Mean delay	1.06	1.06	1.06	1.06	1.06	1.06	1.06	1.06	1.06	1.06
Maximum delay	3	3	3	3	3	3	3	3	3	3

Testfile = airland3.txt

alpha	0.2	0.4	0.6	0.8	1.0	1.2	1.4	1.6	1.8	2.0
Mean delay	2.70	2.70	2.70	2.70	2.70	2.70	2.70	2.70	2.70	2.70
Maximum delay	14	14	14	14	14	14	14	14	14	14

4.2.2 Late approach 1, early approach II

Testfile = airland1.txt

alpha	0.2	0.4	0.6	0.8	1.0	1.2	1.4	1.6	1.8	2.0
Mean delay	-3.7	-3.7	-3.7	-3.7	-3.7	-3.7	-3.7	-3.7	-3.7	-3.7
Maximum delay	2	2	2	2	2	2	2	2	2	2

Testfile = airland2.txt

alpha	0.2	0.4	0.6	0.8	1.0	1.2	1.4	1.6	1.8	2.0
Mean delay	-6.4	-6.4	-6.4	-6.4	-6.4	-6.4	-6.4	-6.4	-6.4	-6.4
Maximum delay	3	3	3	3	3	3	3	3	3	3

Testfile = airland3.txt

alpha	0.2	0.4	0.6	0.8	1.0	1.2	1.4	1.6	1.8	2.0
Mean delay	-18.2	-18.2	-18.2	-18.2	-18.2	-18.2	-18.2	-18.2	-18.2	-18.2
Maximum delay	5	5	5	5	5	5	5	5	5	5

4.2.3 Late approach 1, early approach III

Testfile = airland1.txt

alpha	0.2	0.4	0.6	0.8	1.0	1.2	1.4	1.6	1.8	2.0
Mean delay	-3.7	-3.7	-3.7	-3.7	-3.7	-3.7	-3.7	-3.7	-3.7	-3.7
Maximum delay	2	2	2	2	2	2	2	2	2	2

Testfile = airland2.txt

alpha	0.2	0.4	0.6	0.8	1.0	1.2	1.4	1.6	1.8	2.0
Mean delay	-14	-14	-14	-14	-14	-14	-14	-14	-14	-14
Maximum delay	3	3	3	3	3	3	3	3	3	3

Testfile = airland3.txt

alpha	0.2	0.4	0.6	0.8	1.0	1.2	1.4	1.6	1.8	2.0
Mean delay	-21	-21	-21	-21	-21	-21	-21	-21	-21	-21
Maximum delay	10	10	10	10	10	10	10	10	10	10

4.3 Approach 2

For each test in this section the results are listed in a table with the different values for i listed in the upper row, the calculated values for mean delay in row 2 and the maximum delay presented in row 3.

4.3.1 Late approach 2, early approach I

Testfile = airland1.txt

i	3	4	5	6	7	8	9	10
Mean delay	1.50	1.50	1.50	1.50	1.50	1.50	1.50	1.50
Maximum delay	2	2	2	2	2	2	2	2

Testfile = airland2.txt

i	3	4	5	6	7	8	9	10
Mean delay	1.06	1.06	1.06	1.06	1.06	1.06	1.06	1.06
Maximum delay	3	3	3	3	3	3	3	3

Testfile = airland3.txt

i	3	4	5	6	7	8	9	10
Mean delay	2.70	2.70	2.70	2.70	2.70	2.70	2.70	2.70
Maximum delay	14	14	14	14	14	14	14	14

4.3.2 Late approach 2, early approach II

Testfile = airland1.txt

i	3	4	5	6	7	8	9	10
Mean delay	-3.70	-3.70	-3.70	-3.70	-3.70	-3.70	-3.70	-3.70
Maximum delay	2	2	2	2	2	2	2	2

Testfile = airland2.txt

i	3	4	5	6	7	8	9	10
Mean delay	-6.40	-6.40	-6.40	-6.40	-6.40	-6.40	-6.40	-6.40
Maximum delay	3	3	3	3	3	3	3	3

Testfile = airland3.txt

i	3	4	5	6	7	8	9	10
Mean delay	-18.2	-18.2	-18.2	-18.2	-18.2	-18.2	-18.2	-18.2
Maximum delay	5	5	5	5	5	5	5	5

4.3.3 Late approach 2, early approach III

Testfile = airland1.txt

i	3	4	5	6	7	8	9	10
Mean delay	-3.70	-3.70	-3.70	-3.70	-3.70	-3.70	-3.70	-3.70
Maximum delay	2	2	2	2	2	2	2	2

Testfile = airland2.txt

i	3	4	5	6	7	8	9	10
Mean delay	-14.0	-14.0	-14.0	-14.0	-14.0	-14.0	-14.0	-14.0
Maximum delay	3	3	3	3	3	3	3	3

Testfile = airland3.txt

i	3	4	5	6	7	8	9	10
Mean delay	-21.0	-21.0	-21.0	-21.0	-21.0	-21.0	-21.0	-21.0
Maximum delay	10	10	10	10	10	10	10	10

4.4 Approach 3

For each test in this section the results are listed in a table with the different values for t listed in the upper row, the calculated values for mean delay in row 2 and the maximum delay presented in row 3.

4.4.1 Late approach 3, early approach I

Testfile = airland1.txt

t	20	30	40	50	60	80	100
Mean delay	1.50	1.50	1.50	1.50	1.50	1.50	1.50
Maximum delay	2	2	2	2	2	2	2

Testfile = airland2.txt

t	20	30	40	50	60	80	100
Mean delay	1.06	1.06	1.06	1.06	1.06	1.06	1.06
Maximum delay	3	3	3	3	3	3	3

Testfile = airland3.txt

t	20	30	40	50	60	80	100
Mean delay	2.70	2.70	2.70	2.70	2.70	2.70	2.70
Maximum delay	14	14	14	14	14	14	14

4.4.2 Late approach 3, early approach II

Testfile = airland1.txt

t	20	30	40	50	60	80	100
Mean delay	-3.70	-3.70	-3.70	-3.70	-3.70	-3.70	-3.70
Maximum delay	2	2	2	2	2	2	2

Testfile = airland2.txt

t	20	30	40	50	60	80	100
Mean delay	-6.40	-6.40	-6.40	-6.40	-6.40	-6.40	-6.40
Maximum delay	3	3	3	3	3	3	3

Testfile = airland3.txt

t	20	30	40	50	60	80	100
Mean delay	-18.2	-18.2	-18.2	-18.2	-18.2	-18.2	-18.2
Maximum delay	5	5	5	5	5	5	5

4.4.3 Late approach 3, early approach III

Testfile = airland1.txt

t	20	30	40	50	60	80	100
Mean delay	-3.70	-3.70	-3.70	-3.70	-3.70	-3.70	-3.70
Maximum delay	2	2	2	2	2	2	2

Testfile = airland2.txt

t	20	30	40	50	60	80	100
Mean delay	-14.0	-14.0	-14.0	-14.0	-14.0	-14.0	-14.0
Maximum delay	3	3	3	3	3	3	3

Testfile = airland3.txt

t	20	30	40	50	60	80	100
Mean delay	-21.0	-21.0	-21.0	-21.0	-21.0	-21.0	-21.0
Maximum delay	10	10	10	10	10	10	10

4.5 Approach 4

Since the results from approach 2 with different values for i all were the same it makes no sence doing the test for approach 4.

4.6 Freeze time

Testfile = airland1.txt, freeze = 3.

Approach:	approach 1	approach 2	approach 13
Mean delay	-0.7000	-0.7000	-0.7000
Maximum delay	5	5	5

Testfile = airland1.txt, freeze = 5

Approach:	approach 1	approach 2	approach 13
Mean delay	1.300	1.300	1.300
Maximum delay	7	7	7

Testfile = airland1.txt, freeze = 10

Approach:	approach 1	approach 2	approach 13
Mean delay	6.300	6.300	6.300
Maximum delay	12	12	12

Testfile = airland2.txt, freeze = 3.

Approach:	approach 1	approach 2	approach 3
Mean delay	-14.2000	-14.2000	-14.2000
Maximum delay	3	3	3

Testfile = airland2.txt, freeze = 5

Approach:	approach 1	approach 2	approach 3
Mean delay	-14.2000	-14.2000	-14.2000
Maximum delay	3	3	3

Testfile = airland2.txt, freeze = 10

Approach:	approach 1	approach 2	approach 3
Mean delay	-14.2000	-14.2000	-14.2000
Maximum delay	3	3	3

Testfile = airland3.txt, freeze = 3.

Approach:	approach 1	approach 2	approach 13
Mean delay	-23.8500	-23.8500	-23.8500
Maximum delay	5	5	5

Testfile = airland3.txt, freeze = 5

Approach:	approach 1	approach 2	approach 13
Mean delay	-23.8500	-23.8500	-23.8500
Maximum delay	5	5	5

Testfile = airland3.txt, freeze = 10

Approach:	approach 1	approach 2	approach 13
Mean delay	-23.8500	-23.8500	-23.8500
Maximum delay	5	5	5

4.7 Discussion

The results shows that the late approaches gives the same results. To see if this can be changed additional tests are performed. These tests will all use the early approach III, since this gave the best results compared with early approach I & II.

The first approach is to remove all of the separation constraints to see if they were the reason to the obtained results. Then the tests was conducted with approach 1-3. The results from these tests shows that the different approaches still gives the same results and that every aircraft lands at its earliest landing time.

This indicates that the appearance time of the aircraft are the reasons for the results being equal in the different approaches. The next test conducted had 2 aircraft with the same appearance time. This change was made to see if this would give different results for the different approaches. Unfortunately the results are still equal.

The next change made, was giving the aircraft that inflicted the reschedule a zero penalty for landing besides it's target landing time. Still the results for the different approaches were simular.

One of the main problem for dealing with these dynamic tests are the apperance times for the aircraft. As long as these are static, results will probarbly be simular. A new approach with the apperance time being decided after each rescheduling is tested. In this test there is 80% chance that the apperance time is according to the scheduled landing time and 10% that the aircraft is late and 10% chance that the aircraft is early. If the aircraft apperance time is set early or late it is set randomly earlier or late from between 0 and 20 time-intervals. This did not give any difference in the results from the different approaches. Based on these test it seems that the window time for each aircraft is so small when rescheduling that the approach does not matter. As it can be observed in the next subsection which give an example of how one of the approches are rescheduling when the aircraft approaches the airport, it is only the aircraft causing the reschedule that is rescheduled. As long as this is the case all the approaches will give the same results since they all use the same heuristic. The tests with different freeze time changed the results but this change was made for each of the late approaches.

The java files for these tests are listed in appendix A.3.

4.8 Example

Outline of late approach 1 combined with early approach III

Testfile: airland1.txt

alpha = 1.0

Static schedule

- Aircraft: 2, scheduled landing time: 98
- Aircraft: 3, scheduled landing time: 106
- Aircraft: 4, scheduled landing time: 121
- Aircraft: 5, scheduled landing time: 129
- Aircraft: 6, scheduled landing time: 137
- Aircraft: 7, scheduled landing time: 145
- Aircraft: 0, scheduled landing time: 160
- Aircraft: 8, scheduled landing time: 175
- Aircraft: 9, scheduled landing time: 184
- Aircraft: 1, scheduled landing time: 258

1. reschedule time = 14
   - Aircraft: 2, scheduled landing time: 100
   - Aircraft: 3, scheduled landing time: 106
   - Aircraft: 4, scheduled landing time: 121
   - Aircraft: 5, scheduled landing time: 129
   - Aircraft: 6, scheduled landing time: 137
   - Aircraft: 7, scheduled landing time: 145
   - Aircraft: 0, scheduled landing time: 160
   - Aircraft: 8, scheduled landing time: 175
   - Aircraft: 9, scheduled landing time: 184

2. reschedule time = 21

- Aircraft: 3, scheduled landing time: 108
- Aircraft: 4, scheduled landing time: 121
- Aircraft: 5, scheduled landing time: 129
- Aircraft: 6, scheduled landing time: 137
- Aircraft: 7, scheduled landing time: 145
- Aircraft: 0, scheduled landing time: 160
- Aircraft: 8, scheduled landing time: 175
- Aircraft: 9, scheduled landing time: 184

3. reschedule time = 45
   - Aircraft: 5, scheduled landing time: 131
   - Aircraft: 6, scheduled landing time: 137
   - Aircraft: 7, scheduled landing time: 145
   - Aircraft: 0, scheduled landing time: 160
   - Aircraft: 8, scheduled landing time: 175
   - Aircraft: 9, scheduled landing time: 184
4. reschedule time = 49
   - Aircraft: 6, scheduled landing time: 139
   - Aircraft: 7, scheduled landing time: 145
   - Aircraft: 0, scheduled landing time: 160
   - Aircraft: 8, scheduled landing time: 175
   - Aircraft: 9, scheduled landing time: 184
5. reschedule time = 51
   - Aircraft: 7, scheduled landing time: 147
   - Aircraft: 0, scheduled landing time: 160
   - Aircraft: 8, scheduled landing time: 175
   - Aircraft: 9, scheduled landing time: 184

6. reschedule time = 54
   - Aircraft: 0, scheduled landing time: 162
   - Aircraft: 8, scheduled landing time: 175
   - Aircraft: 9, scheduled landing time: 184
7. reschedule time = 60
   - Aircraft: 8, scheduled landing time: 177
   - Aircraft: 9, scheduled landing time: 184
8. reschedule time = 85
   - Aircraft: 9, scheduled landing time: 185
9. reschedule time = 120
   - Aircraft: 1, scheduled landing time: 206

alpha = 0.2

Chapter 5

Conclusion

5.1 Achivements

A complete program for solving the aircraft landing problem has been implemented in Java. This program can be used to calculate feasable solutions for a given problem with aircraft landings. It is possible to calculate these solution for both the static and the dynamic problem. The test file containing the aircraft can be with special seperation constraints for each pair of individual aircraft or with seperation constraints regarding the types of aircraft involved.

5.2 Problems

A large problem in testing different approaches for solving the dynamic problem is how to create en optimal solution when the apperance time for the future aircraft are unknown. All the calculated results shows that the apperance time is the factor dictating the solution. As long as the apperance time cannnot be estimated before the aircraft starts it's the final landing sequence, the different approaches does not create an unique solutions. If the aircraft give their current position to the persons in charge of the final schedule at some predefined points before starting the landing sequence the apperance time could be estimated much better and the number of reschedulings be decreased.

5.3 Results

The results shows very clearly that the late approaches computes the same results. These results are simular to the results found by the FCFS scheme used which schedules an aircraft at it's earliest possible feasable landing time. Changing the freeze time and thereby changing the periode before the scheduled landing time, where the aircraft is guaranteeing not to be

rescheduled, creates new solutions, these solutions are shared by the different approaches.

For the different approaches on how to deal with early aircraft, it gave a significant increase in the results regarding average delays and it did not have a negetiv impact on the maximum delays, when an early aircraft was allowed to be rescheduled equally with the other aircraft instead of using one of the other two approches for this.

5.4 Recommendations

For continuing the work in dynamic scheduling of aircraft landings, trying to retrieve other test files preferrably from an airport were the aircraft have been rescheduled until their landing time. Might give a clear indication on how to select the aircraft from the static schedule to be rescheduled.

Since the problem being rescheduled are very tight and seldom very large an algorithm guaranteing the optimal solution might be able to find this optimal solution within the amount of time that an airtraffic-controller will tolerate.

If the final population heuristic used in this report should be optimized, a parallel execution of the restarts could increase the computationel time when used in a multi threaded enviroment.

5.5 Usefulness

The algorithms for the dynamic problem does not perform better than a simple FCFS solution. Therefore the created algorithm cannot be recomended for use by air-traffic controllers as a support tool at this stage. Using the FCFS solution for securing that all the seperation constraint are forfilled can be a usefull support tool, especially since the computation time is very low.

Bibliography

[1] Frederic S. Hillier and Gerald J. Lieberman. *Introduction to Operations Research, 8th edition.* McGraw - Hill International Edition, 1994.

[2] Y.M. Sharaiha J.E. Beasley, M. Krishnamoorthy and D. Abramson. Scheduling aircraft landings - the static case. *Transportation Science*, 34, 2000.

[3] J. Soander J.E. Beasley and P. Havelock. Scheduling aircraft landings at londow heathrow using a population heuristic. *Journal of the Operation Research Society*, 52, 2001.

[4] H. Pinol and J.E. Beasley. Scatter search and bionomic algorithms for the aircraft landing problem. *European Journal of Operational Research*, 171, 2006.

[5] John S. Greenwood Jason A.D. Atkin, Edmund K. Burke and Dale Reeson. A meta-heuristic approach to aircraft departure scheduling at london heathrow airport. *electronic proceedings of 9th International Conference on Computer-Aided Scheduling of Public Transport (CASPT), San Diego, California, 9-11 August 2004.*, 2004.

[6] Y.M. Sharaiha J.E. Beasley, M. Krishnamoorthy and D. Abramson. Displacement problem and dynamically sceduling aircraft landings. *Journal of the Operation Research Society*, 55, 2004.

[7] `http://people.brunel.ac.uk/~mastjjb/jeb/orlib/airlandinfo.html`.

Appendix A

Appendix

A.1 Test files

A.1.1 data.txt

```
0 300 500 4
300 350 600 4
50 400 500 4
150 350 550 3
200 350 550 5
```

A.1.2 data2.txt

```
0 300 500 4
300 350 600 4
50 400 500 4
150 350 550 3
200 350 550 5
400 550 900 4
500 650 1200 5
700 650 1100 3
900 900 1400 3
800 750 1200 4
```

A.1.3 airland1.txt

```
10 10
54 129 155 559 10.00 10.00
99999 3 15 15 15 15 15 15 15 15
120 195 258 744 10.00 10.00
3 99999 15 15 15 15 15 15 15 15
14 89 98 510 30.00 30.00
15 15 99999 8 8 8 8 8 8 8
21 96 106 521 30.00 30.00
15 15 8 99999 8 8 8 8 8 8
35 110 123 555 30.00 30.00
15 15 8 8 99999 8 8 8 8 8
45 120 135 576 30.00 30.00
15 15 8 8 8 99999 8 8 8 8
49 124 138 577 30.00 30.00
15 15 8 8 8 8 99999 8 8 8
51 126 140 573 30.00 30.00
15 15 8 8 8 8 8 99999 8 8
60 135 150 591 30.00 30.00
15 15 8 8 8 8 8 8 99999 8
85 160 180 657 30.00 30.00
15 15 8 8 8 8 8 8 8 99999
```

A.1.4 airland2.txt

```
15 10
54 129 155 559 10.00 10.00
99999 3 15 15 15 15 15 15 15 3 3 15 15 3
115 190 250 732 10.00 10.00
3 99999 15 15 15 15 15 15 15 3 3 15 15 3
9 84 93 501 30.00 30.00
15 15 99999 8 8 8 8 8 8 15 15 8 8 15
14 89 98 509 30.00 30.00
15 15 8 99999 8 8 8 8 8 8 15 15 8 8 15
25 100 111 536 30.00 30.00
15 15 8 8 99999 8 8 8 8 8 15 15 8 8 15
32 107 120 552 30.00 30.00
15 15 8 8 8 99999 8 8 8 8 15 15 8 8 15
34 109 121 550 30.00 30.00
15 15 8 8 8 8 99999 8 8 8 15 15 8 8 15
34 109 120 544 30.00 30.00
15 15 8 8 8 8 8 99999 8 8 15 15 8 8 15
40 115 128 557 30.00 30.00
15 15 8 8 8 8 8 8 99999 8 15 15 8 8 15
59 134 151 610 30.00 30.00
15 15 8 8 8 8 8 8 8 99999 15 15 8 8 15
191 266 341 837 10.00 10.00
3 3 15 15 15 15 15 15 15 15 99999 3 15 15 3
176 251 313 778 10.00 10.00
3 3 15 15 15 15 15 15 15 15 3 99999 15 15 3
85 160 181 674 30.00 30.00
15 15 8 8 8 8 8 8 8 8 15 15 99999 8 15
77 152 171 637 30.00 30.00
15 15 8 8 8 8 8 8 8 8 15 15 8 99999 15
201 276 342 815 10.00 10.00
3 3 15 15 15 15 15 15  15 15 3 3 15 15 99999
```

A.1.5 airland3.txt

```
20 10
0 75 82 486 30.00 30.00
99999 15 15 8 15 8 15 8 8 8 8 8 15 15 15 15 15 15 8 8
82 157 197 628 10.00 10.00
15 99999 3 15 3 15 3 15 15 15 15 15 3 3 3 3 3 3 15 15
59 134 160 561 10.00 10.00
15 3 99999 15 3 15 3 15 15 15 15 15 3 3 3 3 3 3 15 15
28 103 117 565 30.00 30.00
8 15 15 99999 15 8 15 8 8 8 8 8 15 15 15 15 15 15 8 8
126 201 261 735 10.00 10.00
15 3 3 15 99999 15 3 15 15 15 15 15 3 3 3 3 3 3 15 15
20 95 106 524 30.00 30.00
8 15 15 8 15 99999 15 8 8 8 8 8 15 15 15 15 15 15 8 8
110 185 229 664 10.00 10.00
15 3 3 15 3 15 99999 15 15 15 15 15 3 3 3 3 3 3 15 15
23 98 108 523 30.00 30.00
8 15 15 8 15 8 15 99999 8 8 8 8 15 15 15 15 15 15 8 8
42 117 132 578 30.00 30.00
8 15 15 8 15 8 15 8 99999 8 8 8 15 15 15 15 15 15 8 8
42 117 130 569 30.00 30.00
8 15 15 8 15 8 15 8 8 99999 8 8 15 15 15 15 15 15 8 8
57 132 149 615 30.00 30.00
8 15 15 8 15 8 15 8 8 8 99999 8 15 15 15 15 15 15 8 8
39 114 126 551 30.00 30.00
8 15 15 8 15 8 15 8 8 8 8 99999 15 15 15 15 15 15 8 8
186 261 336 834 10.00 10.00
15 3 3 15 3 15 3 15 15 15 15 15 99999 3 3 3 3 3 15 15
175 250 316 790 10.00 10.00
15 3 3 15 3 15 3 15 15 15 15 15 3 99999 3 3 3 3 15 15
139 214 258 688 10.00 10.00
15 3 3 15 3 15 3 15 15 15 15 15 3 3 99999 3 3 3 15 15
235 310 409 967 10.00 10.00
15 3 3 15 3 15 3 15 15 15 15 15 3 3 3 99999 3 3 15 15
194 269 338 818 10.00 10.00
15 3 3 15 3 15 3 15 15 15 15 15 3 3 3 3 99999 3 15 15
162 237 287 726 10.00 10.00
15 3 3 15 3 15 3 15 15 15 15 15 3 3 3 3 3 3 99999 15 15
69 144 160 607 30.00 30.00
8 15 15 8 15 8 15 8 8 8 8 8 15 15 15 15 15 15 99999 8
76 151 169 624 30.00 30.00
8 15 15 8 15 8 15 8 8 8 8 8 15 15 15 15 15 15 8 99999
```

A.1.6 airland4.txt

```
20 35
7 82 92 510 30.00 30.00
99999 8 15 15 8 8 8 8 8 15 15 8 8 15 8 8 8 8 8 15
9 84 93 509 30.00 30.00
8 99999 15 15 8 8 8 8 8 15 15 8 8 15 8 8 8 8 8 15
75 150 183 599 10.00 10.00
15 15 99999 3 15 15 15 15 15 3 3 15 15 3 15 15 15 15 15 3
129 204 270 760 10.00 10.00
15 15 3 99999 15 15 15 15 15 3 3 15 15 3 15 15 15 15 15 3
14 89 98 510 30.00 30.00
8 8 15 15 99999 8 8 8 8 15 15 8 8 15 8 8 8 8 8 15
29 104 117 552 30.00 30.00
8 8 15 15 8 99999 8 8 8 15 15 8 8 15 8 8 8 8 8 15
30 105 118 550 30.00 30.00
8 8 15 15 8 8 99999 8 8 15 15 8 8 15 8 8 8 8 8 15
30 105 116 542 30.00 30.00
8 8 15 15 8 8 8 99999 8 15 15 8 8 15 8 8 8 8 8 15
26 101 112 528 30.00 30.00
8 8 15 15 8 8 8 8 99999 15 15 8 8 15 8 8 8 8 8 15
146 221 280 742 10.00 10.00
15 15 3 3 15 15 15 15 15 99999 3 15 15 3 15 15 15 15 15 3
157 232 295 766 10.00 10.00
15 15 3 3 15 15 15 15 15 3 99999 15 15 3 15 15 15 15 15 3
63 138 156 630 30.00 30.00
8 8 15 15 8 8 8 8 8 15 15 99999 8 15 8 8 8 8 8 15
48 123 137 573 30.00 30.00
8 8 15 15 8 8 8 8 8 8 15 15 8 99999 15 8 8 8 8 8 15
160 235 291 746 10.00 10.00
15 15 3 3 15 15 15 15 15 3 3 15 15 99999 15 15 15 15 15 3
78 153 174 663 30.00 30.00
8 8 15 15 8 8 8 8 8 15 15 8 8 15 99999 8 8 8 8 15
65 140 156 609 30.00 30.00
8 8 15 15 8 8 8 8 8 8 15 15 8 8 15 8 99999 8 8 8 15
76 151 170 642 30.00 30.00
8 8 15 15 8 8 8 8 8 8 15 15 8 8 15 8 8 99999 8 8 15
76 151 169 634 30.00 30.00
8 8 15 15 8 8 8 8 8 8 15 15 8 8 15 8 8 8 99999 8 15
75 150 168 627 30.00 30.00
8 8 15 15 8 8 8 8 8 8 15 15 8 8 15 8 8 8 8 99999 15
211 286 357 840 10.00 10.00
15 15 3 3 15 15 15 15 15 3 3 15 15 3 15 15 15 15 15 99999
```

A.1.7 airland9.txt

```
100 720
1 601 908 2401 1.45 1.10
90 90 113 113 90 90 113 90 135 113 90 90 113 113 90 113 90 90 90 90 90 113 90
     135 113 113 90 113 113 113 90 135 90 90 113 113 90 90 113 90 90 135 90 90
     113 113 113 90 113 113 113 90 113 113 113 113 113 90 90 90 90 113 113 113
     113 90 90 113 113 113 113 113 135 113 113 90 135 90 90 113 90 113 90 113 113
      90 113 113 90 90 90 90 90 113 90 113 113 90 113 90
125 725 1027 2525 1.82 1.56
90 90 113 113 90 90 113 90 135 113 90 90 113 113 90 113 90 90 90 90 90 113 90
     135 113 113 90 113 113 113 90 135 90 90 113 113 90 90 113 90 90 135 90 90
     113 113 113 90 113 113 113 90 113 113 113 113 113 90 90 90 90 113 113 113
     113 90 90 113 113 113 113 113 135 113 113 90 135 90 90 113 90 113 90 113 113
      90 113 113 90 90 90 90 90 113 90 113 113 90 113 90
328 928 1378 2728 1.40 1.85
68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
      68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
     68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
     68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
374 974 1502 2774 1.76 1.27
68 68 90 68 68 68 90 68 90 68 68 68 90 68 68 68 68 68 68 68 68 68 68 90 90 68 68
      68 68 68 68 90 68 68 68 90 68 68 68 68 68 90 68 68 68 90 90 68 68 68 68 68
     90 68 68 90 90 68 68 68 68 68 68 90 90 90 68 68 68 90 90 68 68 90 90 68 68 90
     68 68 90 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 90 68
456 1056 1618 2856 1.67 1.55
90 90 113 113 90 90 113 90 135 113 90 90 113 113 90 113 90 90 90 90 90 113 90
     135 113 113 90 113 113 113 90 135 90 90 113 113 90 90 113 90 90 135 90 90
     113 113 113 90 113 113 113 90 113 113 113 113 113 90 90 90 90 113 113 113
     113 90 90 113 113 113 113 113 135 113 113 90 135 90 90 113 90 113 90 113 113
      90 113 113 90 90 90 90 90 113 90 113 113 90 113 90
519 1119 1235 2919 1.39 1.63
90 90 113 113 90 90 113 90 135 113 90 90 113 113 90 113 90 90 90 90 90 113 90
     135 113 113 90 113 113 113 90 135 90 90 113 113 90 90 113 90 90 135 90 90
     113 113 113 90 113 113 113 90 113 113 113 113 113 90 90 90 90 113 113 113
     113 90 90 113 113 113 113 113 135 113 113 90 135 90 90 113 90 113 90 113 113
      90 113 113 90 90 90 90 90 113 90 113 113 90 113 90
548 1148 1520 2948 1.94 1.22
68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
      68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
     68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
     68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
595 1195 1401 2995 1.59 1.82
90 90 113 113 90 90 113 90 135 113 90 90 113 113 90 113 90 90 90 90 90 113 90
     135 113 113 90 113 113 113 90 135 90 90 113 113 90 90 113 90 90 135 90 90
     113 113 113 90 113 113 113 90 113 113 113 113 113 90 90 90 90 113 113 113
     113 90 90 113 113 113 113 113 135 113 113 90 135 90 90 113 90 113 90 113 113
      90 113 113 90 90 90 90 90 113 90 113 113 90 113 90
601 1201 1584 3001 1.02 1.33
68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
      68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
     68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
     68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
618 1218 1343 3018 1.59 1.88
68 68 90 68 68 68 90 68 90 68 68 68 90 68 68 68 68 68 68 68 68 68 68 90 90 68 68
      68 68 68 68 90 68 68 68 90 68 68 68 68 68 90 68 68 68 90 90 68 68 68 68 68
     90 68 68 90 90 68 68 68 68 68 68 90 90 90 68 68 68 90 90 68 68 90 90 68 68 90
     68 68 90 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 90 68
624 1224 1787 3024 1.62 1.15
90 90 113 113 90 90 113 90 135 113 90 90 113 113 90 113 90 90 90 90 90 113 90
     135 113 113 90 113 113 113 90 135 90 90 113 113 90 90 113 90 90 135 90 90
     113 113 113 90 113 113 113 90 113 113 113 113 113 90 90 90 90 113 113 113
     113 90 90 113 113 113 113 113 135 113 113 90 135 90 90 113 90 113 90 113 113
      90 113 113 90 90 90 90 90 113 90 113 113 90 113 90
688 1288 1700 3088 1.13 1.76
90 90 113 113 90 90 113 90 135 113 90 90 113 113 90 113 90 90 90 90 90 113 90
     135 113 113 90 113 113 113 90 135 90 90 113 113 90 90 113 90 90 135 90 90
     113 113 113 90 113 113 113 90 113 113 113 113 113 90 90 90 90 113 113 113
     113 90 90 113 113 113 113 113 135 113 113 90 135 90 90 113 90 113 90 113 113
      90 113 113 90 90 90 90 90 113 90 113 113 90 113 90
1013 1613 1724 3413 1.66 1.23
68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
      68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
     68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
     68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
1071 1671 1907 3471 1.61 1.72
68 68 90 68 68 68 90 68 90 68 68 68 90 68 68 68 68 68 68 68 68 68 68 90 90 68 68
      68 68 68 68 90 68 68 68 90 68 68 68 68 68 90 68 68 68 90 90 68 68 68 68 68
     90 68 68 90 90 68 68 68 68 68 68 90 90 90 68 68 68 90 90 68 68 90 90 68 68 90
     68 68 90 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 90 68
1100 1700 2234 3500 1.78 1.02
90 90 113 113 90 90 113 90 135 113 90 90 113 113 90 113 90 90 90 90 90 113 90
     135 113 113 90 113 113 113 90 135 90 90 113 113 90 90 113 90 90 135 90 90
```

```
     113 113 113 90 113 113 113 90 113 113 113 113 113 90 90 90 90 113 113 113
     113 90 90 113 113 113 113 113 135 113 113 90 135 90 90 113 90 113 90 113 113
      90 113 113 90 90 90 90 90 113 90 113 113 90 113 90
1201 1801 2003 3601 1.09 1.73
68 68 90 68 68 68 90 68 90 68 68 68 90 68 68 68 68 68 68 68 68 68 68 90 90 68 68
      68 68 68 68 90 68 68 68 90 68 68 68 68 68 90 68 68 68 90 90 68 68 68 68 68
     90 68 68 90 90 68 68 68 68 68 90 90 90 68 68 68 90 90 68 68 90 90 68 68 90
     68 68 90 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 90 68
1467 2067 2169 3867 1.85 1.69
90 90 113 113 90 90 113 90 135 113 90 90 113 113 90 113 90 90 90 90 90 113 90
     135 113 113 90 113 113 113 90 135 90 90 113 113 90 90 113 90 90 135 90 90
     113 113 113 90 113 113 113 90 113 113 113 113 113 90 90 90 90 113 113 113
     113 90 90 113 113 113 113 113 135 113 113 90 135 90 90 113 90 113 90 113 113
      90 113 113 90 90 90 90 90 113 90 113 113 90 113 90
1491 2091 2215 3891 1.89 1.17
90 90 113 113 90 90 113 90 135 113 90 90 113 113 90 113 90 90 90 90 90 113 90
     135 113 113 90 113 113 113 90 135 90 90 113 113 90 90 113 90 90 135 90 90
     113 113 113 90 113 113 113 90 113 113 113 113 113 90 90 90 90 113 113 113
     113 90 90 113 113 113 113 113 135 113 113 90 135 90 90 113 90 113 90 113 113
      90 113 113 90 90 90 90 90 113 90 113 113 90 113 90
1642 2242 2781 4042 1.30 1.08
90 90 113 113 90 90 113 90 135 113 90 90 113 113 90 113 90 90 90 90 90 113 90
     135 113 113 90 113 113 113 90 135 90 90 113 113 90 90 113 90 90 135 90 90
     113 113 113 90 113 113 113 90 113 113 113 113 113 90 90 90 90 113 113 113
     113 90 90 113 113 113 113 113 135 113 113 90 135 90 90 113 90 113 90 113 113
      90 113 113 90 90 90 90 90 113 90 113 113 90 113 90
1740 2340 2436 4140 1.67 1.02
90 90 113 113 90 90 113 90 135 113 90 90 113 113 90 113 90 90 90 90 90 113 90
     135 113 113 90 113 113 113 90 135 90 90 113 113 90 90 113 90 90 135 90 90
     113 113 113 90 113 113 113 90 113 113 113 113 113 90 90 90 90 113 113 113
     113 90 90 113 113 113 113 113 135 113 113 90 135 90 90 113 90 113 90 113 113
      90 113 113 90 90 90 90 90 113 90 113 113 90 113 90
2019 2619 3111 4419 1.02 1.29
90 90 113 113 90 90 113 90 135 113 90 90 113 113 90 113 90 90 90 90 90 113 90
     135 113 113 90 113 113 113 90 135 90 90 113 113 90 90 113 90 90 135 90 90
     113 113 113 90 113 113 113 90 113 113 113 113 113 90 90 90 90 113 113 113
     113 90 90 113 113 113 113 113 135 113 113 90 135 90 90 113 90 113 90 113 113
      90 113 113 90 90 90 90 90 113 90 113 113 90 113 90
2105 2705 2967 4505 1.26 1.09
68 68 90 68 68 68 90 68 90 68 68 68 90 68 68 68 68 68 68 68 68 68 68 90 90 68 68
      68 68 68 68 90 68 68 68 90 68 68 68 68 68 90 68 68 68 90 90 68 68 68 68 68
     90 68 68 90 90 68 68 68 68 68 90 90 90 68 68 68 90 90 68 68 90 90 68 68 90
     68 68 90 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 90 68
2126 2726 2971 4526 1.70 1.50
90 90 113 113 90 90 113 90 135 113 90 90 113 113 90 113 90 90 90 90 90 113 90
     135 113 113 90 113 113 113 90 135 90 90 113 113 90 90 113 90 90 135 90 90
     113 113 113 90 113 113 113 90 113 113 113 113 113 90 90 90 90 113 113 113
     113 90 90 113 113 113 113 113 135 113 113 90 135 90 90 113 90 113 90 113 113
      90 113 113 90 90 90 90 90 113 90 113 113 90 113 90
2203 2803 2918 4603 1.08 1.90
68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
      68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
     68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
     68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
2282 2882 3258 4682 1.58 1.83
68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
      68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
     68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
     68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
2646 3246 3728 5046 1.08 1.87
68 68 90 68 68 68 90 68 90 68 68 68 90 68 68 68 68 68 68 68 68 68 68 90 90 68 68
      68 68 68 68 90 68 68 68 90 68 68 68 68 68 90 68 68 68 90 90 68 68 68 68 68
     90 68 68 90 90 68 68 68 68 68 68 90 90 90 68 68 68 90 90 68 68 90 90 68 68 90
     68 68 90 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 90 68
2735 3335 3468 5135 1.14 1.29
90 90 113 113 90 90 113 90 135 113 90 90 113 113 90 113 90 90 90 90 90 113 90
     135 113 113 90 113 113 113 90 135 90 90 113 113 90 90 113 90 90 135 90 90
     113 113 113 90 113 113 113 90 113 113 113 113 113 90 90 90 90 113 113 113
     113 90 90 113 113 113 113 113 135 113 113 90 135 90 90 113 90 113 90 113 113
      90 113 113 90 90 90 90 90 113 90 113 113 90 113 90
2757 3357 3750 5157 1.20 1.10
68 68 90 68 68 68 90 68 90 68 68 68 90 68 68 68 68 68 68 68 68 68 68 90 90 68 68
      68 68 68 68 90 68 68 68 90 68 68 68 68 68 90 68 68 68 90 90 68 68 68 68 68
     90 68 68 90 90 68 68 68 68 68 90 90 90 68 68 68 90 90 68 68 90 90 68 68 90
     68 68 90 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 90 68
2768 3368 3647 5168 1.55 1.78
68 68 90 68 68 68 90 68 90 68 68 68 90 68 68 68 68 68 68 68 68 68 68 90 90 68 68
      68 68 68 68 90 68 68 68 90 68 68 68 68 68 90 68 68 68 90 90 68 68 68 68 68
     90 68 68 90 90 68 68 68 68 68 90 90 90 68 68 68 90 90 68 68 90 90 68 68 90
     68 68 90 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 90 68
2828 3428 3495 5228 1.20 1.44
68 68 90 68 68 68 90 68 90 68 68 68 90 68 68 68 68 68 68 68 68 68 68 90 90 68 68
      68 68 68 68 90 68 68 68 90 68 68 68 68 68 90 68 68 68 90 90 68 68 68 68 68
     90 68 68 90 90 68 68 68 68 68 90 90 90 68 68 68 90 90 68 68 90 90 68 68 90
```

```
     68 68 90 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 90 68
2863 3463 3579 5263 1.27 1.74
90 90 113 113 90 90 113 90 135 113 90 90 113 113 90 113 90 90 90 90 90 113 90
     135 113 113 90 113 113 113 90 135 90 90 113 113 90 90 113 90 90 135 90 90
     113 113 113 90 113 113 113 90 113 113 113 113 113 90 90 90 90 113 113 113
     113 90 90 113 113 113 113 113 135 113 113 90 135 90 90 113 90 113 90 113 113
      90 113 113 90 90 90 90 90 113 90 113 113 90 113 90
2895 3495 4079 5295 1.97 1.78
68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
      68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
     68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
     68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
2944 3544 3764 5344 1.38 1.04
90 90 113 113 90 90 113 90 135 113 90 90 113 113 90 113 90 90 90 90 90 113 90
     135 113 113 90 113 113 113 90 135 90 90 113 113 90 90 113 90 90 135 90 90
     113 113 113 90 113 113 113 90 113 113 113 113 113 90 90 90 90 113 113 113
     113 90 90 113 113 113 113 113 135 113 113 90 135 90 90 113 90 113 90 113 113
      90 113 113 90 90 90 90 90 113 90 113 113 90 113 90
3023 3623 3749 5423 1.66 1.52
90 90 113 113 90 90 113 90 135 113 90 90 113 113 90 113 90 90 90 90 90 113 90
     135 113 113 90 113 113 113 90 135 90 90 113 113 90 90 113 90 90 135 90 90
     113 113 113 90 113 113 113 90 113 113 113 113 113 90 90 90 90 113 113 113
     113 90 90 113 113 113 113 113 135 113 113 90 135 90 90 113 90 113 90 113 113
      90 113 113 90 90 90 90 90 113 90 113 113 90 113 90
3032 3632 3854 5432 1.88 1.99
68 68 90 68 68 68 90 68 90 68 68 68 90 68 68 68 68 68 68 68 68 68 68 90 90 68 68
      68 68 68 68 90 68 68 68 90 68 68 68 68 68 90 68 68 68 90 90 68 68 68 68 68
     90 68 68 90 90 68 68 68 68 68 68 90 90 90 68 68 68 90 90 68 68 90 90 68 68 90
     68 68 90 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 90 68
3123 3723 4273 5523 1.56 1.85
68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
      68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
     68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
     68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
3590 4190 4469 5990 1.20 1.75
90 90 113 113 90 90 113 90 135 113 90 90 113 113 90 113 90 90 90 90 90 113 90
     135 113 113 90 113 113 113 90 135 90 90 113 113 90 90 113 90 90 135 90 90
     113 113 113 90 113 113 113 90 113 113 113 113 113 90 90 90 90 113 113 113
     113 90 90 113 113 113 113 113 135 113 113 90 135 90 90 113 90 113 90 113 113
      90 113 113 90 90 90 90 90 113 90 113 113 90 113 90
3620 4220 4771 6020 1.94 1.29
90 90 113 113 90 90 113 90 135 113 90 90 113 113 90 113 90 90 90 90 90 113 90
     135 113 113 90 113 113 113 90 135 90 90 113 113 90 90 113 90 90 135 90 90
     113 113 113 90 113 113 113 90 113 113 113 113 113 90 90 90 90 113 113 113
     113 90 90 113 113 113 113 113 135 113 113 90 135 90 90 113 90 113 90 113 113
      90 113 113 90 90 90 90 90 113 90 113 113 90 113 90
3646 4246 4836 6046 1.99 1.84
68 68 90 68 68 68 90 68 90 68 68 68 90 68 68 68 68 68 68 68 68 68 68 90 90 68 68
      68 68 68 68 90 68 68 68 90 68 68 68 68 68 90 68 68 68 90 90 68 68 68 68 68
     90 68 68 90 90 68 68 68 68 68 90 90 90 68 68 68 90 90 68 68 90 90 68 68 90
     68 68 90 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 90 68
3776 4376 4695 6176 1.30 1.18
90 90 113 113 90 90 113 90 135 113 90 90 113 113 90 113 90 90 90 90 90 113 90
     135 113 113 90 113 113 113 90 135 90 90 113 113 90 90 113 90 90 135 90 90
     113 113 113 90 113 113 113 90 113 113 113 113 113 90 90 90 90 113 113 113
     113 90 90 113 113 113 113 113 135 113 113 90 135 90 90 113 90 113 90 113 113
      90 113 113 90 90 90 90 90 113 90 113 113 90 113 90
3883 4483 4829 6283 1.67 1.81
90 90 113 113 90 90 113 90 135 113 90 90 113 113 90 113 90 90 90 90 90 113 90
     135 113 113 90 113 113 113 90 135 90 90 113 113 90 90 113 90 90 135 90 90
     113 113 113 90 113 113 113 90 113 113 113 113 113 90 90 90 90 113 113 113
     113 90 90 113 113 113 113 113 135 113 113 90 135 90 90 113 90 113 90 113 113
      90 113 113 90 90 90 90 90 113 90 113 113 90 113 90
4294 4894 5400 6694 1.30 1.08
68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
      68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
     68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
     68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
4365 4965 5536 6765 1.20 1.75
90 90 113 113 90 90 113 90 135 113 90 90 113 113 90 113 90 90 90 90 90 113 90
     135 113 113 90 113 113 113 90 135 90 90 113 113 90 90 113 90 90 135 90 90
     113 113 113 90 113 113 113 90 113 113 113 113 113 90 90 90 90 113 113 113
     113 90 90 113 113 113 113 113 135 113 113 90 135 90 90 113 90 113 90 113 113
      90 113 113 90 90 90 90 90 113 90 113 113 90 113 90
4382 4982 5224 6782 1.57 1.72
90 90 113 113 90 90 113 90 135 113 90 90 113 113 90 113 90 90 90 90 90 113 90
     135 113 113 90 113 113 113 90 135 90 90 113 113 90 90 113 90 90 135 90 90
     113 113 113 90 113 113 113 90 113 113 113 113 113 90 90 90 90 113 113 113
     113 90 90 113 113 113 113 113 135 113 113 90 135 90 90 113 90 113 90 113 113
      90 113 113 90 90 90 90 90 113 90 113 113 90 113 90
4421 5021 5443 6821 1.57 1.78
68 68 90 68 68 68 90 68 90 68 68 68 90 68 68 68 68 68 68 68 68 68 68 90 90 68 68
      68 68 68 68 90 68 68 68 90 68 68 68 68 68 90 68 68 68 90 90 68 68 68 68 68
     90 68 68 90 90 68 68 68 68 68 90 90 90 68 68 68 90 90 68 68 90 90 68 68 90
```

```
    68 68 90 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 90 68
4430 5030 5492 6830 1.22 1.35
68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
     68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
    68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
    68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
4545 5145 5373 6945 1.70 1.08
68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
     68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
    68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
    68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
4575 5175 5731 6975 1.99 1.20
90 90 113 113 90 90 113 90 135 113 90 90 113 113 90 113 90 90 90 90 90 113 90
    135 113 113 90 113 113 113 90 135 90 90 113 113 90 90 113 90 90 135 90 90
    113 113 113 90 113 113 113 90 113 113 113 113 113 90 90 90 90 113 113 113
    113 90 90 113 113 113 113 113 135 113 113 90 135 90 90 113 90 113 90 113 113
     90 113 113 90 90 90 90 90 113 90 113 113 90 113 90
4614 5214 5749 7014 1.19 1.37
68 68 90 68 68 68 90 68 90 68 68 68 90 68 68 68 68 68 68 68 68 68 68 90 90 68 68
     68 68 68 68 90 68 68 68 90 68 68 68 68 68 90 68 68 68 90 90 68 68 68 68 68
    90 68 68 90 90 68 68 68 68 68 90 90 90 68 68 68 90 90 68 68 90 90 68 68 90
    68 68 90 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 90 68
4624 5224 5588 7024 1.13 1.55
68 68 90 68 68 68 90 68 90 68 68 68 90 68 68 68 68 68 68 68 68 68 68 90 90 68 68
     68 68 68 68 90 68 68 68 90 68 68 68 68 68 90 68 68 68 90 90 68 68 68 68 68
    90 68 68 90 90 68 68 68 68 68 90 90 90 68 68 68 90 90 68 68 90 90 68 68 90
    68 68 90 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 90 68
4746 5346 5636 7146 1.61 1.92
68 68 90 68 68 68 90 68 90 68 68 68 90 68 68 68 68 68 68 68 68 68 68 90 90 68 68
     68 68 68 68 90 68 68 68 90 68 68 68 68 68 90 68 68 68 90 90 68 68 68 68 68
    90 68 68 90 90 68 68 68 68 68 90 90 90 68 68 68 90 90 68 68 90 90 68 68 90
    68 68 90 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 90 68
4988 5588 5946 7388 1.46 1.74
90 90 113 113 90 90 113 90 135 113 90 90 113 113 90 113 90 90 90 90 90 113 90
    135 113 113 90 113 113 113 90 135 90 90 113 113 90 90 113 90 90 135 90 90
    113 113 113 90 113 113 113 90 113 113 113 113 113 90 90 90 90 113 113 113
    113 90 90 113 113 113 113 113 135 113 113 90 135 90 90 113 90 113 90 113 113
     90 113 113 90 90 90 90 90 113 90 113 113 90 113 90
5091 5691 6145 7491 1.97 1.89
68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
     68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
    68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
    68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
5104 5704 6074 7504 1.75 1.38
68 68 90 68 68 68 90 68 90 68 68 68 90 68 68 68 68 68 68 68 68 68 68 90 90 68 68
     68 68 68 68 90 68 68 68 90 68 68 68 68 68 90 68 68 68 90 90 68 68 68 68 68
    90 68 68 90 90 68 68 68 68 68 90 90 90 68 68 68 90 90 68 68 90 90 68 68 90
    68 68 90 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 90 68
5208 5808 6223 7608 1.61 1.20
68 68 90 68 68 68 90 68 90 68 68 68 90 68 68 68 68 68 68 68 68 68 68 90 90 68 68
     68 68 68 68 90 68 68 68 90 68 68 68 68 68 90 68 68 68 90 90 68 68 68 68 68
    90 68 68 90 90 68 68 68 68 68 90 90 90 68 68 68 90 90 68 68 90 90 68 68 90
    68 68 90 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 90 68
5344 5944 6521 7744 1.89 1.09
68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
     68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
    68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
    68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
5381 5981 6404 7781 1.17 1.50
68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
     68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
    68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
    68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
5502 6102 6455 7902 1.47 1.32
90 90 113 113 90 90 113 90 135 113 90 90 113 113 90 113 90 90 90 90 90 113 90
    135 113 113 90 113 113 113 90 135 90 90 113 113 90 90 113 90 90 135 90 90
    113 113 113 90 113 113 113 90 113 113 113 113 113 90 90 90 90 113 113 113
    113 90 90 113 113 113 113 113 135 113 113 90 135 90 90 113 90 113 90 113 113
     90 113 113 90 90 90 90 90 113 90 113 113 90 113 90
5608 6208 6344 8008 1.44 1.87
90 90 113 113 90 90 113 90 135 113 90 90 113 113 90 113 90 90 90 90 90 113 90
    135 113 113 90 113 113 113 90 135 90 90 113 113 90 90 113 90 90 135 90 90
    113 113 113 90 113 113 113 90 113 113 113 113 113 90 90 90 90 113 113 113
    113 90 90 113 113 113 113 113 135 113 113 90 135 90 90 113 90 113 90 113 113
     90 113 113 90 90 90 90 90 113 90 113 113 90 113 90
5982 6582 6867 8382 1.39 1.89
90 90 113 113 90 90 113 90 135 113 90 90 113 113 90 113 90 90 90 90 90 113 90
    135 113 113 90 113 113 113 90 135 90 90 113 113 90 90 113 90 90 135 90 90
    113 113 113 90 113 113 113 90 113 113 113 113 113 90 90 90 90 113 113 113
    113 90 90 113 113 113 113 113 135 113 113 90 135 90 90 113 90 113 90 113 113
     90 113 113 90 90 90 90 90 113 90 113 113 90 113 90
6242 6842 6972 8642 1.80 1.29
90 90 113 113 90 90 113 90 135 113 90 90 113 113 90 113 90 90 90 90 90 113 90
    135 113 113 90 113 113 113 90 135 90 90 113 113 90 90 113 90 90 135 90 90
```

```
    113 113 113 90 113 113 113 90 113 113 113 113 113 90 90 90 90 113 113 113
    113 90 90 113 113 113 113 113 135 113 113 90 135 90 90 113 90 113 90 113 113
     90 113 113 90 90 90 90 90 113 90 113 113 90 113 90
6568 7168 7439 8968 1.80 1.54
68 68 90 68 68 68 90 68 90 68 68 68 90 68 68 68 68 68 68 68 68 68 68 90 90 68 68
     68 68 68 68 90 68 68 68 90 68 68 68 68 68 90 68 68 68 90 90 68 68 68 68 68
    90 68 68 90 90 68 68 68 68 68 90 90 90 68 68 68 90 90 68 68 90 90 68 68 90
    68 68 90 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 90 68
6640 7240 7635 9040 1.07 1.46
68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
     68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
    68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
    68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
6908 7508 7824 9308 1.82 1.27
68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
     68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
    68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
    68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
7032 7632 8162 9432 1.55 1.80
68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
     68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
    68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
    68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
7035 7635 8124 9435 1.49 1.50
90 90 113 113 90 90 113 90 135 113 90 90 113 113 90 113 90 90 90 90 90 113 90
    135 113 113 90 113 113 113 90 135 90 90 113 113 90 90 113 90 90 135 90 90
    113 113 113 90 113 113 113 90 113 113 113 113 113 90 90 90 90 113 113 113
    113 90 90 113 113 113 113 113 135 113 113 90 135 90 90 113 90 113 90 113 113
     90 113 113 90 90 90 90 90 113 90 113 113 90 113 90
7125 7725 8080 9525 1.37 1.51
90 90 113 113 90 90 113 90 135 113 90 90 113 113 90 113 90 90 90 90 90 113 90
    135 113 113 90 113 113 113 90 135 90 90 113 113 90 90 113 90 90 135 90 90
    113 113 113 90 113 113 113 90 113 113 113 113 113 90 90 90 90 113 113 113
    113 90 90 113 113 113 113 113 135 113 113 90 135 90 90 113 90 113 90 113 113
     90 113 113 90 90 90 90 90 113 90 113 113 90 113 90
7174 7774 8149 9574 1.65 1.26
68 68 90 68 68 68 90 68 90 68 68 68 90 68 68 68 68 68 68 68 68 68 68 90 90 68 68
     68 68 68 68 90 68 68 68 90 68 68 68 68 68 90 68 68 68 90 90 68 68 68 68 68
    90 68 68 90 90 68 68 68 68 68 90 90 90 68 68 68 90 90 68 68 90 90 68 68 90
    68 68 90 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 90 68
7471 8071 8291 9871 1.80 1.85
68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
     68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
    68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
    68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
7552 8152 8349 9952 1.73 1.02
68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
     68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
    68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
    68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
7644 8244 8715 10044 1.96 1.53
68 68 90 68 68 68 90 68 90 68 68 68 90 68 68 68 68 68 68 68 68 68 68 90 90 68 68
     68 68 68 68 90 68 68 68 90 68 68 68 68 68 90 68 68 68 90 90 68 68 68 68 68
    90 68 68 90 90 68 68 68 68 68 90 90 90 68 68 68 90 90 68 68 90 90 68 68 90
    68 68 90 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 90 68
7750 8350 8455 10150 1.74 1.53
68 68 90 68 68 68 90 68 90 68 68 68 90 68 68 68 68 68 68 68 68 68 68 90 90 68 68
     68 68 68 68 90 68 68 68 90 68 68 68 68 68 90 68 68 68 90 90 68 68 68 68 68
    90 68 68 90 90 68 68 68 68 68 90 90 90 68 68 68 90 90 68 68 90 90 68 68 90
    68 68 90 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 90 68
8023 8623 8903 10423 1.77 1.61
68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
     68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
    68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
    68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
8401 9001 9293 10801 1.54 1.29
68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
     68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
    68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
    68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
8765 9365 9508 11165 1.48 1.65
68 68 90 68 68 68 90 68 90 68 68 68 90 68 68 68 68 68 68 68 68 68 68 90 90 68 68
     68 68 68 68 90 68 68 68 90 68 68 68 68 68 90 68 68 68 90 90 68 68 68 68 68
    90 68 68 90 90 68 68 68 68 68 90 90 90 68 68 68 90 90 68 68 90 90 68 68 90
    68 68 90 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 90 68
8969 9569 9988 11369 1.62 1.82
90 90 113 113 90 90 113 90 135 113 90 90 113 113 90 113 90 90 90 90 90 113 90
    135 113 113 90 113 113 113 90 135 90 90 113 113 90 90 113 90 90 135 90 90
    113 113 113 90 113 113 113 90 113 113 113 113 113 90 90 90 90 113 113 113
    113 90 90 113 113 113 113 113 135 113 113 90 135 90 90 113 90 113 90 113 113
     90 113 113 90 90 90 90 90 113 90 113 113 90 113 90
9030 9630 9796 11430 1.73 1.53
68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
     68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
```

```
     68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
     68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
9220 9820 10300 11620 1.76 1.99
90 90 113 113 90 90 113 90 135 113 90 90 113 113 90 113 90 90 90 90 90 113 90
     135 113 113 90 113 113 113 90 135 90 90 113 113 90 90 113 90 90 135 90 90
     113 113 113 90 113 113 113 90 113 113 113 113 113 90 90 90 90 113 113 113
     113 90 90 113 113 113 113 113 135 113 113 90 135 90 90 113 90 113 90 113 113
      90 113 113 90 90 90 90 90 113 90 113 113 90 113 90
9222 9822 10108 11622 1.76 1.83
90 90 113 113 90 90 113 90 135 113 90 90 113 113 90 113 90 90 90 90 90 113 90
     135 113 113 90 113 113 113 90 135 90 90 113 113 90 90 113 90 90 135 90 90
     113 113 113 90 113 113 113 90 113 113 113 113 113 90 90 90 90 113 113 113
     113 90 90 113 113 113 113 113 135 113 113 90 135 90 90 113 90 113 90 113 113
      90 113 113 90 90 90 90 90 113 90 113 113 90 113 90
9519 10119 10666 11919 1.61 1.75
68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
      68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
     68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
     68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
9552 10152 10704 11952 1.72 1.84
90 90 113 113 90 90 113 90 135 113 90 90 113 113 90 113 90 90 90 90 90 113 90
     135 113 113 90 113 113 113 90 135 90 90 113 113 90 90 113 90 90 135 90 90
     113 113 113 90 113 113 113 90 113 113 113 113 113 90 90 90 90 113 113 113
     113 90 90 113 113 113 113 113 135 113 113 90 135 90 90 113 90 113 90 113 113
      90 113 113 90 90 90 90 90 113 90 113 113 90 113 90
9788 10388 10779 12188 1.97 1.26
68 68 90 68 68 68 90 68 90 68 68 68 90 68 68 68 68 68 68 68 68 68 68 90 90 68 68
      68 68 68 68 90 68 68 68 90 68 68 68 68 68 68 90 68 68 68 90 90 68 68 68 68 68
     90 68 68 90 90 68 68 68 68 68 90 90 90 68 68 68 90 90 68 68 90 90 68 68 90
     68 68 90 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 90 68
9797 10397 10944 12197 1.21 1.96
90 90 113 113 90 90 113 90 135 113 90 90 113 113 90 113 90 90 90 90 90 113 90
     135 113 113 90 113 113 113 90 135 90 90 113 113 90 90 113 90 90 135 90 90
     113 113 113 90 113 113 113 90 113 113 113 113 113 90 90 90 90 113 113 113
     113 90 90 113 113 113 113 113 135 113 113 90 135 90 90 113 90 113 90 113 113
      90 113 113 90 90 90 90 90 113 90 113 113 90 113 90
9811 10411 10943 12211 1.05 1.19
68 68 90 68 68 68 90 68 90 68 68 68 90 68 68 68 68 68 68 68 68 68 68 90 90 68 68
      68 68 68 68 90 68 68 68 90 68 68 68 68 68 90 68 68 68 90 90 68 68 68 68 68
     90 68 68 90 90 68 68 68 68 68 68 90 90 90 68 68 68 90 90 68 68 90 90 68 68 90
     68 68 90 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 90 68
9844 10444 10515 12244 1.06 1.87
68 68 90 68 68 68 90 68 90 68 68 68 90 68 68 68 68 68 68 68 68 68 68 90 90 68 68
      68 68 68 68 90 68 68 68 90 68 68 68 68 68 68 90 68 68 68 90 90 68 68 68 68 68
     90 68 68 90 90 68 68 68 68 68 68 90 90 90 68 68 68 90 90 68 68 90 90 68 68 90
     68 68 90 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 90 68
10141 10741 11112 12541 1.74 1.88
90 90 113 113 90 90 113 90 135 113 90 90 113 113 90 113 90 90 90 90 90 113 90
     135 113 113 90 113 113 113 90 135 90 90 113 113 90 90 113 90 90 135 90 90
     113 113 113 90 113 113 113 90 113 113 113 113 113 90 90 90 90 113 113 113
     113 90 90 113 113 113 113 113 135 113 113 90 135 90 90 113 90 113 90 113 113
      90 113 113 90 90 90 90 90 113 90 113 113 90 113 90
10216 10816 11288 12616 1.39 1.63
68 68 90 68 68 68 90 68 90 68 68 68 90 68 68 68 68 68 68 68 68 68 68 90 90 68 68
      68 68 68 68 90 68 68 68 90 68 68 68 68 68 90 68 68 68 90 90 68 68 68 68 68
     90 68 68 90 90 68 68 68 68 68 90 90 90 68 68 68 90 90 68 68 90 90 68 68 90
     68 68 90 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 90 68
10258 10858 11071 12658 1.14 1.10
68 68 90 68 68 68 90 68 90 68 68 68 90 68 68 68 68 68 68 68 68 68 68 90 90 68 68
      68 68 68 68 90 68 68 68 90 68 68 68 68 68 90 68 68 68 90 90 68 68 68 68 68
     90 68 68 90 90 68 68 68 68 68 68 90 90 90 68 68 68 90 90 68 68 90 90 68 68 90
     68 68 90 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 90 68
10330 10930 11353 12730 1.49 1.76
90 90 113 113 90 90 113 90 135 113 90 90 113 113 90 113 90 90 90 90 90 113 90
     135 113 113 90 113 113 113 90 135 90 90 113 113 90 90 113 90 90 135 90 90
     113 113 113 90 113 113 113 90 113 113 113 113 113 90 90 90 90 113 113 113
     113 90 90 113 113 113 113 113 135 113 113 90 135 90 90 113 90 113 90 113 113
      90 113 113 90 90 90 90 90 113 90 113 113 90 113 90
10443 11043 11335 12843 1.91 1.85
90 90 113 113 90 90 113 90 135 113 90 90 113 113 90 113 90 90 90 90 90 113 90
     135 113 113 90 113 113 113 90 135 90 90 113 113 90 90 113 90 90 135 90 90
     113 113 113 90 113 113 113 90 113 113 113 113 113 90 90 90 90 113 113 113
     113 90 90 113 113 113 113 113 135 113 113 90 135 90 90 113 90 113 90 113 113
      90 113 113 90 90 90 90 90 113 90 113 113 90 113 90
10623 11223 11428 13023 1.00 1.50
90 90 113 113 90 90 113 90 135 113 90 90 113 113 90 113 90 90 90 90 90 113 90
     135 113 113 90 113 113 113 90 135 90 90 113 113 90 90 113 90 90 135 90 90
     113 113 113 90 113 113 113 90 113 113 113 113 113 90 90 90 90 113 113 113
     113 90 90 113 113 113 113 113 135 113 113 90 135 90 90 113 90 113 90 113 113
      90 113 113 90 90 90 90 90 113 90 113 113 90 113 90
10679 11279 11713 13079 1.84 1.84
90 90 113 113 90 90 113 90 135 113 90 90 113 113 90 113 90 90 90 90 90 113 90
     135 113 113 90 113 113 113 90 135 90 90 113 113 90 90 113 90 90 135 90 90
     113 113 113 90 113 113 113 90 113 113 113 113 113 90 90 90 90 113 113 113
```

```
     113 90 90 113 113 113 113 113 135 113 113 90 135 90 90 113 90 113 90 113 113
      90 113 113 90 90 90 90 90 113 90 113 113 90 113 90
10970 11570 11959 13370 1.53 1.95
90 90 113 113 90 90 113 90 135 113 90 90 113 113 90 113 90 90 90 90 90 113 90
     135 113 113 90 113 113 113 90 135 90 90 113 113 90 90 113 90 90 135 90 90
     113 113 113 90 113 113 113 90 113 113 113 113 113 90 90 90 90 113 113 113
     113 90 90 113 113 113 113 113 135 113 113 90 135 90 90 113 90 113 90 113 113
      90 113 113 90 90 90 90 90 113 90 113 113 90 113 90
11089 11689 12035 13489 1.02 1.78
68 68 90 68 68 68 90 68 90 68 68 68 90 68 68 68 68 68 68 68 68 68 68 90 90 68 68
      68 68 68 68 90 68 68 68 90 68 68 68 68 68 90 68 68 68 90 90 68 68 68 68 68
     90 68 68 90 90 68 68 68 68 68 90 90 90 68 68 68 90 90 68 68 90 90 68 68 90
     68 68 90 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 90 68
11137 11737 12031 13537 1.73 1.06
90 90 113 113 90 90 113 90 135 113 90 90 113 113 90 113 90 90 90 90 90 113 90
     135 113 113 90 113 113 113 90 135 90 90 113 113 90 90 113 90 90 135 90 90
     113 113 113 90 113 113 113 90 113 113 113 113 113 90 90 90 90 113 113 113
     113 90 90 113 113 113 113 113 135 113 113 90 135 90 90 113 90 113 90 113 113
      90 113 113 90 90 90 90 90 113 90 113 113 90 113 90
11342 11942 12393 13742 1.44 1.68
68 68 90 68 68 68 90 68 90 68 68 68 90 68 68 68 68 68 68 68 68 68 68 90 90 68 68
      68 68 68 68 90 68 68 68 90 68 68 68 68 68 90 68 68 68 90 90 68 68 68 68 68
     90 68 68 90 90 68 68 68 68 68 90 90 90 68 68 68 90 90 68 68 90 90 68 68 90
     68 68 90 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 90 68
11524 12124 12655 13924 1.79 1.03
68 68 90 68 68 68 90 68 90 68 68 68 90 68 68 68 68 68 68 68 68 68 68 90 90 68 68
      68 68 68 68 90 68 68 68 90 68 68 68 68 68 90 68 68 68 90 90 68 68 68 68 68
     90 68 68 90 90 68 68 68 68 68 90 90 90 68 68 68 90 90 68 68 90 90 68 68 90
     68 68 90 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 90 68
11536 12136 12387 13936 1.99 1.26
90 90 113 113 90 90 113 90 135 113 90 90 113 113 90 113 90 90 90 90 90 113 90
     135 113 113 90 113 113 113 90 135 90 90 113 113 90 90 113 90 90 135 90 90
     113 113 113 90 113 113 113 90 113 113 113 113 113 90 90 90 90 113 113 113
     113 90 90 113 113 113 113 113 135 113 113 90 135 90 90 113 90 113 90 113 113
      90 113 113 90 90 90 90 90 113 90 113 113 90 113 90
11638 12238 12550 14038 1.03 1.84
68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
      68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
     68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
     68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68
11723 12323 12691 14123 1.71 1.52
90 90 113 113 90 90 113 90 135 113 90 90 113 113 90 113 90 90 90 90 90 113 90
     135 113 113 90 113 113 113 90 135 90 90 113 113 90 90 113 90 90 135 90 90
     113 113 113 90 113 113 113 90 113 113 113 113 113 90 90 90 90 113 113 113
     113 90 90 113 113 113 113 113 135 113 113 90 135 90 90 113 90 113 90 113 113
      90 113 113 90 90 90 90 90 113 90 113 113 90 113 90
```

A.2 Java files for the static tests

A.2.1 Test1.java

```
package sorting;

import model.Solution;

public class Test1 {

        /**
         * @param args
         */
        public static void main(String[] args) {

                GenerateInitialPopulation gip = new GenerateInitialPopulation
                    (100, "data.txt");
                PopulationHeuristic ph = new PopulationHeuristic(gip.
                    getPopulation(),"target");

                int j =0;

                Solution sol;

                while(j<1000){
                        sol = ph.createChild();
                        ph.populationReplacement(sol);
                        if(sol.getUnfitness()==0){
                                break;
                        }
                        j++;
                }

                System.out.println(j);
        }
}
```

A.2.2 Test2.java

```
 package sorting;

import model.Solution;

public class Test2 {

        /**
         * @param args
         */
        public static void main(String[] args) {
                Solution best = new Solution();
                Solution test = new Solution();

                GenerateInitialPopulation gip = new GenerateInitialPopulation
                    (100, "data.txt");
                PopulationHeuristic ph;

                int j =0, k=0;


                Solution sol;

                while(k++<10){
                        ph = new PopulationHeuristic(gip.getPopulation(),"
                            scheduled");

                        j =0;
                        if(gip.getBestSolution().getUnfitness()==0)
                                best = gip.getBestSolution();
                        else
                                best = new Solution();

                        while(j<5000){
                                sol = ph.createChild();
                                ph.populationReplacement(sol);
                                if(sol.getUnfitness()==0){
                                        if(sol.getFitness()>best.getFitness()){
                                                best = sol;
                                                j=0;
                                        }
                                }
                                j++;
                        }
                        if(best.getFitness()>test.getFitness())
                                test = best;
                }

                if(test.getUnfitness()!=0)
                        System.out.println("No feasable solutions");
                else{
                        System.out.println("Best solution: "+test.getFitness());
                        //System.out.println("k: "+k);
                        System.out.println(test.toString());
                }
        }
}
```

A.2.3 Test3.java

```
package sorting;

import java.util.ArrayList;
import java.util.Iterator;

import model.Aircraft;
import model.AircraftExt;
import model.ImportAircraftExt;
import model.Solution;

public class Test3 {
        private ImportAircraftExt ia;
        private ArrayList<AircraftExt> list;
        private Solution best;
        private int populationSize, iterations;

        public Test3(String filename, int populationSize, int iterations){
                this.populationSize = populationSize;
                this.iterations = iterations;
                best = new Solution();
                ia = new ImportAircraftExt(filename);
                list = ia.getAircraftList();
                firstIteration();
        }

        public void firstIteration(){
                Solution bestGip = new Solution();
                Solution temp = new Solution();

                int k=0;
                while(k++<10){
                        GenerateInitialPopulation gip = new
                            GenerateInitialPopulation(populationSize, list);
                        PopulationHeuristic ph = new PopulationHeuristic(gip.
                            getPopulation(),"scheduled");

                        int j =0;

                        Solution sol;
                        bestGip = gip.getBestSolution();

                        if(bestGip.getFitness()<best.getFitness())
                                best = bestGip;

                        while(j<iterations){
                                sol = ph.createChild();
                                ph.populationReplacementExt(sol);
                                if(sol.getUnfitness()==0){
                                        temp = sol;
                                        if(sol.getFitness()<best.getFitness()){
                                                best = sol;
                                                j=0;
                                        }
                                }
                                j++;
                        }
                }
                if(best.getUnfitness()==0)
                        tightening(best);
                else
                        tightening(temp);
        }

        public void tightening(Solution sol){
                Iterator<Aircraft> it = sol.getList().iterator();

                while(it.hasNext()){
                        AircraftExt air = (AircraftExt)it.next();
                        if(air.getScheduledLandingTime()>air.getTargetLandingTime
                            ()){
                                air.setLatestLandingTime(air.
                                    getScheduledLandingTime());
                        }
                        else if(air.getScheduledLandingTime()<air.
                            getTargetLandingTime()){
                                air.setEarliestLandindgTime(air.
                                    getScheduledLandingTime());
                        }
                }
                iterate(sol);
        }
```

```
public void iterate(Solution solution){
        Solution bestGip = new Solution();
        Solution temp = new Solution();
        int k=0;

        while(k++<10){
                GenerateInitialPopulation gip = new
                    GenerateInitialPopulation(populationSize, solution.
                    getListExt());
                PopulationHeuristic ph = new PopulationHeuristic(gip.
                    getPopulation(),"scheduled");

                int j =0;

                Solution sol;
                bestGip = gip.getBestSolution();

                if(bestGip.getFitness()<best.getFitness() && bestGip.
                    getUnfitness()==0)
                        best = bestGip;

                while(j<iterations){
                        sol = ph.createChild();
                        ph.populationReplacementExt(sol);
                        if(sol.getUnfitness()==0){
                                temp = sol;
                                if(sol.getFitness()<best.getFitness()){
                                        best = sol;
                                        j=0;
                                }
                        }
                        j++;
                }
        }
        if(best.getFitness()<solution.getFitness() && best.getUnfitness()
            ==0){
                tightening(best);
        }
        else if(temp.getFitness()<solution.getFitness()){
                tightening(temp);
        }
        else{
                //System.out.println("Best solution: "+best.getFitness())
                    ;
                //System.out.println(best.toString());
                mutate(best);
        }
}

public void mutate(Solution sol){
        AircraftExt one;
        AircraftExt two;
        int dist, temp;
        int earliest = 0;

        for(int i=0; i<sol.getList().size()-1;i++){
                one = (AircraftExt)sol.getList().get(i);
                two = (AircraftExt)sol.getList().get(i+1);
                dist = two.getScheduledLandingTime()-one.
                    getScheduledLandingTime();
                if(dist>one.getSeperation(i+1)){
                        if(one.getScheduledLandingTime()>one.
                            getTargetLandingTime()){
                                if(one.getTargetLandingTime()>earliest){
                                        one.setScheduledLandingTime(one.
                                            getTargetLandingTime());
                                }
                                else if(earliest < one.
                                    getScheduledLandingTime()){
                                        one.setScheduledLandingTime(
                                            earliest);
                                }
                        }
                        else if(one.getScheduledLandingTime()<one.
                            getTargetLandingTime()){
                                temp = one.getScheduledLandingTime()+dist
                                    -one.getSeperation(i+1);
                                one.setScheduledLandingTime(Math.min(one.
                                    getTargetLandingTime(), temp));
                        }
                }
                earliest = one.getScheduledLandingTime()+one.
                    getSeperation(i+1);
```

```
                }

                Evaluation eval = new Evaluation();
                sol.setFitness(eval.fintnessLinearExt(sol));
                System.out.println("Mutated solution: "+sol.getFitness());
                System.out.println(sol.toString());
        }

        public static void main(String[] args){
                Test3 te = new Test3("airland3.txt", 500, 10000);
        }
}
```

A.3 Java files for the dynamic tests

A.3.1 DynamicTest1.java

```
package sorting;

import java.util.ArrayList;
import java.util.Iterator;

import model.Aircraft;
import model.AircraftExt;
import model.Solution;

public class DynamicTest1 {
        private double alpha;
        private int max, freeze;
        private FinalHeuristic fh;
        private AircraftExt old=null;

        /**
         * @param args
         */
        public DynamicTest1(double alpha, Solution solution, int freeze){
                max = 15; //The maximin seperation criteria for the test files
                this.alpha = alpha;
                fh = new FinalHeuristic();
                scan(solution);
                this.freeze = freeze;
        }

        //Scans for late or early aircraft
        public void scan(Solution solution){
                ArrayList<AircraftExt> list = solution.getListExt();

                int time=0;
                //Scan for incoming aircraft while there still are aircraft in
                    the air
                while(list.size()>0){
                        for(int i=0;i<list.size();i++){
                                AircraftExt air = list.get(i);
                                if(air.getApperanceTime()-freeze==time){
                                        //System.out.println("Time: "+time);
                                        if(air.getApperanceTime()<air.
                                             getScheduledLandingTime()-86-freeze){
                                                //System.out.println("Early
                                                     aircraft: "+air.getNumber());
                                                selectAircraftEarly(list,air);
                                        }
                                        else if(air.getApperanceTime()>air.
                                             getScheduledLandingTime()-86-freeze){
                                                //System.out.println("Late
                                                     aircraft: "+air.getNumber());
                                                selectAircraftLate(list,air);
                                        }
                                }
                                else if(air.getScheduledLandingTime()==time){
                                        //Aircraft landed
                                        list.remove(air);
                                        old=air;
                                        System.out.println(air);
                                }
                        }
                        time++;
                }
        }

        //Selects the aircraft for rescheduling
        public void selectAircraftLate(ArrayList<AircraftExt> airList, Aircraft
             air){
                int gap = 0;
                int i = airList.indexOf(air);
                int j = i;

                while(gap < max*alpha && i<airList.size()-1){
                        AircraftExt air1 = ((AircraftExt)airList.get(i+1));
                        AircraftExt air2 = ((AircraftExt)airList.get(i));

                        gap += air1.getScheduledLandingTime()- air2.
                            getScheduledLandingTime()-
                        air2.getSeperation(air1.getNumber());
                        i++;
```

```
                }
                fh.initialize(airList, j, i,freeze);
        }

        public void selectAircraftEarly(ArrayList<AircraftExt> airList,
             AircraftExt air){
                //If an aircraft is early set its scheduled time dependent on the
                     aircraft arriving
                //before it if the seperation constraints are nut fulfilled.
                /*if(airList.size()>1 && airList.indexOf(air)!=0){
                        AircraftExt ae = airList.get(airList.indexOf(air)-1);
                        air.setScheduledLandingTime(Math.max(ae.
                             getScheduledLandingTime()+
                                        ae.getSeperation(air.getNumber()),air.
                                             getScheduledLandingTime()));
                }*/
                //Treat a early aircraft as a late aircraft
                selectAircraftLate(airList, air);

                //Special case for the early aircraft
                /*if(airList.size()>2){
                        int i = airList.indexOf(air);
                        for(int j=0; j<i-1; j++){
                                AircraftExt ae1 = airList.get(j);
                                AircraftExt ae2 = airList.get(j+1);
                                int window = ae2.getScheduledLandingTime()-ae1.
                                     getScheduledLandingTime();
                                int sep1 = ae1.getSeperation(air.getNumber());
                                int sep2 = air.getSeperation(ae2.getNumber());

                                if(window >= sep1+sep2){
                                        if(ae1.getScheduledLandingTime()+sep1>air
                                             .getApperanceTime()+86)
                                                air.setScheduledLandingTime(ae1.
                                                     getScheduledLandingTime()+
                                                     sep1);
                                }
                                else{
                                        air.setScheduledLandingTime(Math.max(ae2.
                                             getScheduledLandingTime()+
                                                     ae2.getSeperation(air.
                                                          getNumber()),air.
                                                          getApperanceTime()
                                                          +86));
                                }
                        }
                }
                else if(airList.size()==1){
                        air.setScheduledLandingTime(Math.max(air.getApperanceTime
                             ()+86,
                                        old.getScheduledLandingTime()+old.
                                             getSeperation(air.getNumber())));
                        airList.remove(air);
                        System.out.println(air+" last");
                }*/
        }

        public static void main(String[] args) {
                // TODO Auto-generated method stub
                Solution sol = new StaticSchedule(3).getSol();
                //System.out.println(sol.getListExt());
                DynamicTest1 dt = new DynamicTest1(1, sol,5);
        }

}
```

A.3.2 DynamicTest2.java

```
package sorting;

import java.util.ArrayList;

import model.Aircraft;
import model.AircraftExt;
import model.Solution;

public class DynamicTest2 {
        private FinalHeuristic fh;
        private int number, freeze;
        private AircraftExt old=null;
        /**
         * @param args
         */
        public DynamicTest2(int number, Solution solution, int freeze){
                this.number = number;
                this.freeze = freeze;
                fh = new FinalHeuristic();
                scan(solution);
        }

        public void scan(Solution solution){
                ArrayList<AircraftExt> list = solution.getListExt();

                int time=0;
                //Scan for incoming aircraft while there still are aircraft in
                    the air
                while(list.size()>0){
                        for(int i=0;i<list.size();i++){
                                AircraftExt air = list.get(i);
                                if(air.getApperanceTime()-freeze==time){
                                        if(air.getApperanceTime()<air.
                                             getScheduledLandingTime()-86-freeze){
                                            //System.out.println("Early
                                                 aircraft: "+air.getNumber());
                                            selectAircraftEarly(list,air);
                                        }
                                        else if(air.getApperanceTime()>air.
                                             getScheduledLandingTime()-86-freeze){
                                            //System.out.println("Late
                                                 aircraft: "+air.getNumber());
                                            selectAircraftLate(list,air);
                                        }
                                }
                                else if(air.getScheduledLandingTime()==time){
                                        //Aircraft landed
                                        list.remove(air);
                                        old=air;
                                        System.out.println(air);
                                }
                        }
                        time++;
                }
        }

//      Selects the aircraft for rescheduling
        public void selectAircraftLate(ArrayList<AircraftExt> airList, Aircraft
             air){
                int j = airList.indexOf(air);
                if(airList.size()>1)
                        fh.initialize(airList, j, j+number, freeze);
                else if(airList.size()==1){
                        air.setScheduledLandingTime(Math.max(air.getApperanceTime
                             ()+86,
                                        old.getScheduledLandingTime()+old.
                                             getSeperation(air.getNumber())));
                        airList.remove(air);
                        System.out.println(air);
                }
        }

        public void selectAircraftEarly(ArrayList<AircraftExt> airList,
             AircraftExt air){
//              If an aircraft is early set its scheduled time dependent on the
    aircraft arriving
                //before it if the seperation constraints are nut fulfilled.
                /*if(airList.size()>1 && airList.indexOf(air)!=0){
                        AircraftExt ae = airList.get(airList.indexOf(air)-1);
                        air.setScheduledLandingTime(Math.max(ae.
                             getScheduledLandingTime()+
```

```
                                        ae.getSeperation(air.getNumber()),air.
                                            getScheduledLandingTime()));
                }*/
                //Treat a early aircraft as a late aircraft
                selectAircraftLate(airList, air);

                //Special case for the early aircraft
                /*if(airList.size()>2){
                        int i = airList.indexOf(air);
                        for(int j=0; j<i-1; j++){
                                AircraftExt ae1 = airList.get(j);
                                AircraftExt ae2 = airList.get(j+1);
                                int window = ae2.getScheduledLandingTime()-ae1.
                                    getScheduledLandingTime();
                                int sep1 = ae1.getSeperation(air.getNumber());
                                int sep2 = air.getSeperation(ae2.getNumber());

                                if(window >= sep1+sep2){
                                        if(ae1.getScheduledLandingTime()+sep1>air
                                            .getApperanceTime()+86)
                                                air.setScheduledLandingTime(ae1.
                                                    getScheduledLandingTime()+
                                                    sep1);
                                }
                                else{
                                        air.setScheduledLandingTime(Math.max(ae2.
                                            getScheduledLandingTime()+
                                                ae2.getSeperation(air.
                                                    getNumber()),air.
                                                    getApperanceTime()
                                                    +86));
                                }
                        }
                }
                else if(airList.size()==1){
                        air.setScheduledLandingTime(Math.max(air.getApperanceTime
                            ()+86,
                                        old.getScheduledLandingTime()+old.
                                            getSeperation(air.getNumber())));
                        airList.remove(air);
                        System.out.println(air+" last");
                }*/
        }
        public static void main(String[] args) {
                // TODO Auto-generated method stub
                Solution sol = new StaticSchedule(1).getSol();
                DynamicTest2 dt = new DynamicTest2(8, sol,20);
        }

}
```

A.3.3 DynamicTest3.java

```
package sorting;

import java.util.ArrayList;

import model.Aircraft;
import model.AircraftExt;
import model.Solution;

public class DynamicTest3 {
        private FinalHeuristic fh;
        private int time, freeze;
        private int t;
        private AircraftExt old=null;
        /**
         * @param args
         */
        public DynamicTest3(int t, Solution solution, int freeze){
                this.t = t;
                this.freeze = freeze;
                fh = new FinalHeuristic();
                scan(solution);
        }

        public void scan(Solution solution){
                ArrayList<AircraftExt> list = solution.getListExt();

                time=0;
                //Scan for incoming aircraft while there still are aircraft in
                    the air
                while(list.size()>0){
                        for(int i=0;i<list.size();i++){
                                AircraftExt air = list.get(i);
                                if(air.getApperanceTime()-freeze==time){
                                        if(air.getApperanceTime()<air.
                                            getScheduledLandingTime()-86-freeze){
                                                //System.out.println("Early
                                                    aircraft: "+air.getNumber());
                                                selectAircraftEarly(list,air);
                                        }
                                        else if(air.getApperanceTime()>air.
                                            getScheduledLandingTime()-86-freeze){
                                                //System.out.println("Late
                                                    aircraft: "+air.getNumber());
                                                selectAircraftLate(list,air);
                                        }
                                }
                                else if(air.getScheduledLandingTime()==time){
                                        //Aircraft landed
                                        list.remove(air);
                                        old=air;
                                        System.out.println(air);
                                }
                        }
                        time++;
                }
        }

//      Selects the aircraft for rescheduling
        public void selectAircraftLate(ArrayList<AircraftExt> airList, Aircraft
            air){
                int i=0;
                while(i<airList.size() && airList.get(i).getScheduledLandingTime
                    ()<time+86+t)
                        i++;
                if(airList.size()>1 && i!=0){
                        fh.initialize(airList, airList.indexOf(air), i,freeze);
                }
                else if(airList.size()==1){
                        air.setScheduledLandingTime(Math.max(air.getApperanceTime
                            ()+86,
                                        old.getScheduledLandingTime()+old.
                                            getSeperation(air.getNumber())));
                        airList.remove(air);
                        System.out.println(air+"␣late");
                }
        }

        public void selectAircraftEarly(ArrayList<AircraftExt> airList,
            AircraftExt air){
                //If an aircraft is early set its scheduled time dependent on the
                     aircraft arriving
```

```
                //before it if the seperation constraints are nut fulfilled.
                /*if(airList.size()>1 && airList.indexOf(air)!=0){
                        AircraftExt ae = airList.get(airList.indexOf(air)-1);
                        air.setScheduledLandingTime(Math.max(ae.
                             getScheduledLandingTime()+
                                        ae.getSeperation(air.getNumber()),air.
                                             getScheduledLandingTime()));
                }*/
                //Treat a early aircraft as a late aircraft
                selectAircraftLate(airList, air);

                //Special case for the early aircraft
                /*if(airList.size()>2){
                        int i = airList.indexOf(air);
                        for(int j=0; j<i-1; j++){
                                AircraftExt ae1 = airList.get(j);
                                AircraftExt ae2 = airList.get(j+1);
                                int window = ae2.getScheduledLandingTime()-ae1.
                                     getScheduledLandingTime();
                                int sep1 = ae1.getSeperation(air.getNumber());
                                int sep2 = air.getSeperation(ae2.getNumber());

                                if(window >= sep1+sep2){
                                        if(ae1.getScheduledLandingTime()+sep1>air
                                             .getApperanceTime()+86)
                                                air.setScheduledLandingTime(ae1.
                                                     getScheduledLandingTime()+
                                                     sep1);
                                }
                                else{
                                        air.setScheduledLandingTime(Math.max(ae2.
                                             getScheduledLandingTime()+
                                                     ae2.getSeperation(air.
                                                          getNumber()),air.
                                                          getApperanceTime()
                                                          +86));
                                }
                        }
                }
                else if(airList.size()==1){
                        air.setScheduledLandingTime(Math.max(air.getApperanceTime
                             ()+86,
                                        old.getScheduledLandingTime()+old.
                                             getSeperation(air.getNumber())));
                        airList.remove(air);
                        System.out.println(air+" last");
                }*/
        }
        public static void main(String[] args) {
                // TODO Auto-generated method stub
                Solution sol = new StaticSchedule(1).getSol();
                DynamicTest3 dt = new DynamicTest3(30, sol,20);
        }
}
```

A.3.4 FinalHeuristic.java

```
package sorting;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.Random;

import model.Aircraft;
import model.AircraftExt;
import model.Solution;

public class FinalHeuristic {
        private int populationSize;
        private int iterations;
        private Solution best;
        private Random rand;

        public FinalHeuristic(){
                populationSize = 500;
                iterations = 1000;
                best = new Solution();
                rand = new Random();
        }

        public void initialize(ArrayList<AircraftExt> airList, int j, int i){
                ArrayList<AircraftExt> list = new ArrayList<AircraftExt>();
                i = Math.min(i,airList.size()-1);

                if(j==i){
                        list.add(airList.get(j));
                }
                else{
                        for(int k=j; k<i; k++){
                                list.add((AircraftExt)airList.get(k));
                        }
                }

                AircraftExt air,airRef;
                int earliest;

                airRef = (AircraftExt)airList.get(i);
                AircraftExt old = airList.get(0);
                if(j!=0){
                        for(int h=0; h<j; h++){
                                if(airList.get(h).getScheduledLandingTime()>old.
                                     getScheduledLandingTime())
                                        old = airList.get(h);
                        }
                        earliest = Math.max(list.get(0).getApperanceTime()+86,
                                        old.getScheduledLandingTime()+old.
                                             getSeperation(list.get(0).getNumber()
                                             ));
                }
                else if(old!=null && old!=airList.get(j)){
                        earliest = Math.max(airList.get(j).getApperanceTime()+86,
                                        old.getScheduledLandingTime()+old.
                                             getSeperation(airList.get(j).
                                             getNumber()));
                }
                else
                        earliest =list.get(0).getApperanceTime()+86;

                list.get(0).setEarliestLandindgTime(earliest);
                list.get(0).setScheduledLandingTime(list.get(0).
                     getEarliestLandindgTime());

                for(int k=1; k<list.size(); k++){
                                air = list.get(k);
                                air.setEarliestLandindgTime(air.
                                     getScheduledLandingTime());
                                air.setTargetLandingTime(air.
                                     getScheduledLandingTime());
                                air.setLatestLandingTime(Math.min(air.
                                     getScheduledLandingTime()+75,
                                                airRef.getScheduledLandingTime()-
                                                     airRef.getSeperation(air.
                                                     getNumber())));
                }
                //System.out.println(list);
                //AircraftExt ae = list.get(0);
                //double a = ae.getPenA();
                //double b = ae.getPenB();
```

```
            //ae.setPenA(0);
            //ae.setPenB(0);
            firstIteration(list);
            //ae.setPenA(a);
            //ae.setPenB(b);
    }

    public void firstIteration(ArrayList<AircraftExt> list){
            Solution bestGip = new Solution();
            Solution temp = new Solution();
            Solution sol;
            best = new Solution();

            int k=0;
            while(k++<10){
                    GenerateInitialPopulation gip = new
                        GenerateInitialPopulation(populationSize, list);
                    PopulationHeuristic ph = new PopulationHeuristic(gip.
                        getPopulation(),"scheduled");

                    int j =0;


                    bestGip = gip.getBestSolution();

                    if(bestGip.getFitness()<best.getFitness()){
                            //best = bestGip;
                            best.setListExt(bestGip.getListExt());
                            best.setFitness(bestGip.getFitness());
                            best.setUnfitness(bestGip.getUnfitness());
                    }

                    while(j<iterations){
                            sol = ph.createChild();
                            ph.populationReplacementExt(sol);
                            if(sol.getUnfitness()==0){
                                    temp.setListExt(sol.getListExt());
                                    if(sol.getFitness()<best.getFitness()){
                                            best.setListExt(sol.getListExt())
                                                ;
                                            best.setFitness(sol.getFitness())
                                                ;
                                            best.setUnfitness(sol.
                                                getUnfitness());
                                            j=0;
                                    }
                            }
                            j++;
                    }
            }
            if(best.getUnfitness()==0)
                    tightening(best);
            else
                    tightening(temp);
    }

    public void tightening(Solution sol){
            Iterator<AircraftExt> it = sol.getListExt().iterator();

            while(it.hasNext()){
                    AircraftExt air = it.next();
                    if(air.getScheduledLandingTime()>air.getTargetLandingTime
                        ()){
                            air.setLatestLandingTime(air.
                                getScheduledLandingTime());
                    }
                    else if(air.getScheduledLandingTime()<air.
                        getTargetLandingTime()){
                            air.setEarliestLandindgTime(air.
                                getScheduledLandingTime());
                    }
            }
            iterate(sol);
    }

    public void iterate(Solution solution){
            Solution bestGip = new Solution();
            Solution temp = new Solution();
            int k=0;

            while(k++<10){
                    GenerateInitialPopulation gip = new
                        GenerateInitialPopulation(populationSize, solution.
                        getListExt());
```

```
                    PopulationHeuristic ph = new PopulationHeuristic(gip.
                        getPopulation(),"scheduled");

                    int j =0;

                    Solution sol;
                    bestGip = gip.getBestSolution();

                    if(bestGip.getFitness()<best.getFitness() && bestGip.
                        getUnfitness()==0)
                            best = bestGip;

                    while(j<iterations){
                            sol = ph.createChild();
                            ph.populationReplacementExt(sol);
                            if(sol.getUnfitness()==0){
                                    temp = sol;
                                    if(sol.getFitness()<best.getFitness()){
                                            best = sol;
                                            j=0;
                                    }
                            }
                            j++;
                    }
            }
            if(best.getFitness()<solution.getFitness() && best.getUnfitness()
                ==0){
                    tightening(best);
            }
            else if(temp.getFitness()<solution.getFitness()){
                    if(temp.getUnfitness()!=0)
                            System.out.println(temp);
                    tightening(temp);
            }
            else{
                    if(best.getUnfitness()!=0)
                            System.out.println(best);
                    mutate(best);
            }
    }

    public void mutate(Solution sol){
            AircraftExt one;
            AircraftExt two;
            int dist, temp;
            int earliest = 0;

            for(int i=0; i<sol.getList().size()-1;i++){
                    one = (AircraftExt)sol.getList().get(i);
                    two = (AircraftExt)sol.getList().get(i+1);
                    dist = two.getScheduledLandingTime()-one.
                        getScheduledLandingTime();
                    if(dist>one.getSeperation(i+1)){
                            if(one.getScheduledLandingTime()>one.
                                getTargetLandingTime()){
                                    if(one.getTargetLandingTime()>earliest){
                                            one.setScheduledLandingTime(one.
                                                getTargetLandingTime());
                                    }
                                    else if(earliest < one.
                                        getScheduledLandingTime()){
                                            one.setScheduledLandingTime(
                                                earliest);
                                    }
                            }
                            else if(one.getScheduledLandingTime()<one.
                                getTargetLandingTime()){
                                    temp = one.getScheduledLandingTime()+dist
                                        -one.getSeperation(i+1);
                                    one.setScheduledLandingTime(Math.min(one.
                                        getTargetLandingTime(), temp));
                            }
                    }
                    earliest = one.getScheduledLandingTime()+one.
                        getSeperation(i+1);
            }

            Evaluation eval = new Evaluation();
            sol.setFitness(eval.fintnessLinearExt(sol));
            sol.setUnfitness(eval.unfitnessExt(sol));
            //System.out.println("Mutated solution: "+sol.getFitness());
            //System.out.println(sol.getListExt());
            //System.out.println("Unfitness: "+sol.getUnfitness());
            app(sol);
```

```
        }

        public void app(Solution sol){
                Iterator it = sol.getListExt().iterator();
                AircraftExt air;
                int early, late;
                while(it.hasNext()){
                        air = (AircraftExt) it.next();
                        int onTime = rand.nextInt(100);
                        if(onTime < 10){
                                early = air.getScheduledLandingTime()-86-rand.
                                    nextInt(21);
                                air.setApperanceTime(early);
                        }
                        else if(onTime > 89){
                                late = air.getScheduledLandingTime()-86+rand.
                                    nextInt(21);
                                air.setApperanceTime(late);
                        }
                        else
                                air.setApperanceTime(air.getScheduledLandingTime
                                    ()-86);
                }
        }
}
```

A.3.5 FCFS.java

```
package sorting;

import java.util.ArrayList;
import java.util.Iterator;

import model.Aircraft;
import model.AircraftExt;
import model.Solution;

public class FCFS {
        private double alpha;
        private int max;
        private FinalHeuristic fh;
        private AircraftExt old=null;
        private int earliest;

        /**
         * @param args
         */
        public FCFS(Solution solution){
                fh = new FinalHeuristic();
                scan(solution);
        }

        //Scans for late or early aircraft
        public void scan(Solution solution){
                ArrayList<AircraftExt> list = solution.getListExt();

                int time=0;
                //Scan for incoming aircraft while there still are aircraft in
                    the air
                while(list.size()>0){
                        for(int i=0;i<list.size();i++){
                                AircraftExt air = list.get(i);
                                if(air.getApperanceTime()==time){
                                        AircraftExt old = list.get(0);
                                        if(i!=0){
                                                for(int h=0; h<i; h++){
                                                        if(list.get(h).
                                                            getScheduledLandingTime
                                                            ()>old.
                                                            getScheduledLandingTime
                                                            ())
                                                                old = list.get(h)
                                                                    ;
                                                }
                                                earliest = Math.max(list.get(i).
                                                    getApperanceTime()+86,
                                                                old.
                                                                    getScheduledLandingTime
                                                                    ()+old.
                                                                    getSeperation
                                                                    (list.get(i).
                                                                    getNumber()))
                                                                    ;
                                        }
                                        else
                                                earliest =list.get(0).
                                                    getApperanceTime()+86;

                                        air.setScheduledLandingTime(earliest);
                                }
                                else if(air.getScheduledLandingTime()==time){
                                        //Aircraft landed
                                        list.remove(air);
                                        old=air;
                                        System.out.println(air);
                                }
                        }
                        time++;
                }
        }


        public static void main(String[] args) {
                // TODO Auto-generated method stub
                Solution sol = new StaticSchedule(3).getSol();
                //System.out.println(sol.getListExt());
                FCFS fc = new FCFS(sol);
        }
```

```
}
```

A.3.6 StaticSchedule.java

```
package sorting;

import java.util.ArrayList;
import java.util.Iterator;

import model.Aircraft;
import model.AircraftExt;
import model.ImportAircraftExt;
import model.ImportAircrafts;
import model.Solution;

public class StaticSchedule {
        Solution sol;

        public StaticSchedule(int i){

                switch(i){
                case 1: sol=one(); break;
                case 2: sol=two(); break;
                case 3: sol=three(); break;
                case 4: sol=four(); break;
                case 9: sol=nine(); break;
                default: System.out.println("Illegal input parameter");
                }
        }

        public Solution one(){
                //Changes due to test
                //ImportAircraftExt ia = new ImportAircraftExt("airland1_0.txt");
                //ImportAircraftExt ia = new ImportAircraftExt("airland1_1.txt");

                //Original import
                ImportAircraftExt ia = new ImportAircraftExt("airland1.txt");

                ArrayList<AircraftExt> al = ia.getAircraftList();

                al.get(0).setScheduledLandingTime(98);
                al.get(1).setScheduledLandingTime(106);
                al.get(2).setScheduledLandingTime(121);
                al.get(3).setScheduledLandingTime(129);
                al.get(4).setScheduledLandingTime(137);
                al.get(5).setScheduledLandingTime(145);
                al.get(6).setScheduledLandingTime(160);
                al.get(7).setScheduledLandingTime(175);
                al.get(8).setScheduledLandingTime(184);
                al.get(9).setScheduledLandingTime(258);

                Solution solution = new Solution();
                solution.setListExt(al);
                solution.sort("apperance");
                return solution;
        }

        public Solution two(){
                ImportAircraftExt ia = new ImportAircraftExt("airland2.txt");

                ArrayList<AircraftExt> al = ia.getAircraftList();

                al.get(0).setScheduledLandingTime(92);
                al.get(1).setScheduledLandingTime(100);
                al.get(2).setScheduledLandingTime(109);
                al.get(3).setScheduledLandingTime(117);
                al.get(4).setScheduledLandingTime(125);
                al.get(5).setScheduledLandingTime(133);
                al.get(6).setScheduledLandingTime(141);
                al.get(7).setScheduledLandingTime(158);
                al.get(8).setScheduledLandingTime(173);
                al.get(9).setScheduledLandingTime(185);
                al.get(10).setScheduledLandingTime(197);
                al.get(11).setScheduledLandingTime(250);
                al.get(12).setScheduledLandingTime(313);
                al.get(13).setScheduledLandingTime(339);
                al.get(14).setScheduledLandingTime(342);

                Solution solution = new Solution();
                solution.setListExt(al);
                solution.sort("apperance");
                return solution;
        }

        public Solution three(){
```

```
                ImportAircraftExt ia = new ImportAircraftExt("airland3.txt");

                ArrayList<AircraftExt> al = ia.getAircraftList();

                al.get(0).setScheduledLandingTime(82);
                al.get(1).setScheduledLandingTime(101);
                al.get(2).setScheduledLandingTime(109);
                al.get(3).setScheduledLandingTime(117);
                al.get(4).setScheduledLandingTime(131);
                al.get(5).setScheduledLandingTime(141);
                al.get(6).setScheduledLandingTime(150);
                al.get(7).setScheduledLandingTime(158);
                al.get(8).setScheduledLandingTime(166);
                al.get(9).setScheduledLandingTime(181);
                al.get(10).setScheduledLandingTime(189);
                al.get(11).setScheduledLandingTime(197);
                al.get(12).setScheduledLandingTime(229);
                al.get(13).setScheduledLandingTime(261);
                al.get(14).setScheduledLandingTime(264);
                al.get(15).setScheduledLandingTime(287);
                al.get(16).setScheduledLandingTime(316);
                al.get(17).setScheduledLandingTime(336);
                al.get(18).setScheduledLandingTime(351);
                al.get(19).setScheduledLandingTime(433);

                Solution solution = new Solution();
                solution.setListExt(al);
                solution.sort("apperance");
                return solution;
        }

        public Solution four(){
                ImportAircraftExt ia = new ImportAircraftExt("airland4.txt");

                ArrayList<AircraftExt> al = ia.getAircraftList();
                Iterator<AircraftExt> it = al.iterator();

                while(it.hasNext()){
                        AircraftExt air = it.next();
                        air.setScheduledLandingTime(air.getTargetLandingTime());
                }

                Solution solution = new Solution();
                solution.setListExt(al);
                solution.sort("apperance");
                return solution;
        }

        public Solution nine(){
                ImportAircraftExt ia = new ImportAircraftExt("airland9.txt");

                ArrayList<AircraftExt> al = ia.getAircraftList();
                Iterator<AircraftExt> it = al.iterator();

                while(it.hasNext()){
                        AircraftExt air = it.next();
                        air.setScheduledLandingTime(air.getTargetLandingTime());
                }

                Solution solution = new Solution();
                solution.setListExt(al);
                solution.sort("apperance");
                return solution;
        }

        public Solution getSol() {
                return sol;
        }
}
```

A.4 Other java files

PopulationHeuristic.java

```
package sorting;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.Random;

import model.Solution;

public class PopulationHeuristic {
        private ArrayList<Solution> population;
        private Random rand;
        private Evaluation eval;
        private String objective;

        public PopulationHeuristic(ArrayList<Solution> population, String
             objective){
                rand = new Random();
                eval = new Evaluation();
                this.population = population;
                this.objective = objective;
        }

        public Solution getRandomParent(){
                //returns a random parent
                if(population.size()==0){
                        return new Solution();
                }
                return population.get(rand.nextInt(population.size()));
        }

        public Solution selectParent(Solution one, Solution two){
                //Return the parent with best fitness value
                if (one.getFitness()>two.getFitness())
                        return one;
                else
                        return two;
        }

        public Solution createChild(){
                //Selects 4 different parants randomly
                Solution one = getRandomParent();
                Solution two = getRandomParent();
                Solution three = getRandomParent();
                Solution four = getRandomParent();

                //Binary tournament, to find the best 2 parents
                Solution parentOne = selectParent(one, two);
                Solution parentTwo = selectParent(three, four);

                //Crossover of the 2 parents
                Solution temp = uniformCrossover(parentOne, parentTwo);

                //Sort the child according to the objective
                temp.sort(objective);

                //Calculate the fitness value
                //int fitness = eval.fintnessLinear(temp);
                //int fitness = eval.fintnessNonLinear(temp);
                int fitness = eval.fintnessLinearExt(temp);

                //Calculate the unfitness value
                //int unfitness = eval.unfitness(temp);
                int unfitness = eval.unfitnessExt(temp);

                //Sets the fitness and unfitness value of the child
                temp.setFitness(fitness);
                temp.setUnfitness(unfitness);

                return temp;
        }

        public Solution uniformCrossover(Solution one, Solution two){
                int i = 0;
                Solution temp = new Solution();
                boolean index;
                one.sort("number");
                two.sort("number");
```

```
            while(i<one.getListExt().size()){
                    index = rand.nextBoolean();
                    if(index)
                            temp.getListExt().add(one.getListExt().get(i));
                    else
                            temp.getListExt().add(two.getListExt().get(i));
                    i++;
            }
            temp.setFitness(eval.fintnessLinearExt(temp));
            temp.setUnfitness(eval.unfitnessExt(temp));
            return temp;
    }

    public void populationReplacement(Solution child){
            ArrayList<Solution> g1 = new ArrayList<Solution>();
            ArrayList<Solution> g2 = new ArrayList<Solution>();
            ArrayList<Solution> g3 = new ArrayList<Solution>();
            ArrayList<Solution> g4 = new ArrayList<Solution>();

            Solution temp;

            Iterator<Solution> it = population.iterator();

            //Sorting the current solutions into groups
            while(it.hasNext()){
                    temp = it.next();
                    if(temp.getFitness()<child.getFitness()){
                            if(temp.getUnfitness()<child.getUnfitness())
                                    g3.add(temp);
                            else
                                    g1.add(temp);
                    }
                    else{
                            if(temp.getUnfitness()<child.getUnfitness())
                                    g4.add(temp);
                            else
                                    g2.add(temp);
                    }
            }

            //remove a solution from the population randomly from the
            //worst group if possible, othervise form the second worst etc.
            if(g1.size()>0)
                    population.remove(g1.get(rand.nextInt(g1.size())));
            else if(g2.size()>0)
                    population.remove(g2.get(rand.nextInt(g2.size())));
            else if(g3.size()>0)
                    population.remove(g3.get(rand.nextInt(g3.size())));
            else
                    population.remove(g4.get(rand.nextInt(g4.size())));

            //add the child to the population
            population.add(child);

    }

    public void populationReplacementExt(Solution child){
            ArrayList<Solution> g1 = new ArrayList<Solution>();
            ArrayList<Solution> g2 = new ArrayList<Solution>();
            ArrayList<Solution> g3 = new ArrayList<Solution>();
            ArrayList<Solution> g4 = new ArrayList<Solution>();

            Solution temp;

            Iterator<Solution> it = population.iterator();

            //Sorting the current solutions into groups
            while(it.hasNext()){
                    temp = it.next();
                    if(Math.abs(temp.getFitness())>Math.abs(child.getFitness
                         ())){
                            if(temp.getUnfitness()<child.getUnfitness())
                                    g3.add(temp);
                            else
                                    g1.add(temp);
                    }
                    else{
                            if(temp.getUnfitness()<child.getUnfitness())
                                    g4.add(temp);
                            else
                                    g2.add(temp);
                    }
            }
```

```
                //remove a solution from the population randomly from the
                //worst group if possible, othervise form the second worst etc.
                if(g1.size()>0)
                        population.remove(g1.get(rand.nextInt(g1.size())));
                else if(g2.size()>0)
                        population.remove(g2.get(rand.nextInt(g2.size())));
                else if(g3.size()>0)
                        population.remove(g3.get(rand.nextInt(g3.size())));
                else if(g4.size()>0)
                        population.remove(g4.get(rand.nextInt(g4.size())));

                //add the child to the population
                population.add(child);

        }

        public int totalFitness(ArrayList<Solution> pop){
                int sum = 0;
                Iterator<Solution> it = pop.iterator();
                while(it.hasNext()){
                        sum += it.next().getFitness();
                }
                return sum;
        }

        public int totalUnfitness(ArrayList<Solution> pop){
                int sum = 0;
                Iterator<Solution> it = pop.iterator();
                while(it.hasNext()){
                        sum += it.next().getUnfitness();
                }
                return sum;
        }


        public ArrayList<Solution> getPopulation() {
                return population;
        }
}
```

A.4.1 GenerateInitialPopulation.java

```
package sorting;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.Random;

import model.Aircraft;
import model.AircraftExt;
import model.ImportAircraftExt;
import model.ImportAircrafts;
import model.Solution;

public class GenerateInitialPopulation {
        private ImportAircraftExt ia;
        private ArrayList<AircraftExt> airList;
        private ArrayList<AircraftExt> airListExt=null;
        private ArrayList<Solution> population;
        private Evaluation eval;
        private Solution bestSolution;

        public GenerateInitialPopulation(int number, String filename){
                ia = new ImportAircraftExt(filename);
                population = new ArrayList<Solution>();
                eval = new Evaluation();
                bestSolution = new Solution();
                airList = ia.getAircraftList();
                createRandomPopulation(number);
        }

        public GenerateInitialPopulation(int number, ArrayList<AircraftExt>
             airListExt){
                population = new ArrayList<Solution>();
                eval = new Evaluation();
                bestSolution = new Solution();
                this.airListExt = airListExt;
                createRandomPopulationExt(number);
        }

        public void createRandomPopulation(int number){
                Solution sol;
                AircraftExt temp, new_aircraft;
                Iterator<AircraftExt> it;
                int window, landing;
                Random rand = new Random();

                for(int i=0; i<number; i++){
                        it = airList.iterator();
                        sol = new Solution();
                        while(it.hasNext()){
                                temp = it.next();
                                window = temp.getLatestLandingTime()-temp.
                                     getEarliestLandindgTime();
                                landing = temp.getEarliestLandindgTime()+rand.
                                     nextInt(window);
                                if(temp.getApperanceTime()!=-1)
                                        new_aircraft = AircraftExt.
                                             createNewAircraftExt(temp);
                                else
                                        new_aircraft = AircraftExt.
                                             createNewAircraftExt(temp);

                                new_aircraft.setScheduledLandingTime(landing);
                                sol.getListExt().add(new_aircraft);
                        }
                        sol.setFitness(eval.fintnessNonLinear(sol));
                        sol.setUnfitness(eval.unfitness(sol));
                        //Finds the initial best solution
                        if(sol.getUnfitness()==0 && sol.getFitness()>bestSolution
                             .getFitness())
                                bestSolution = sol;
                        population.add(sol);
                }
        }

        public void createRandomPopulationExt(int number){
                Solution sol;
                AircraftExt temp, new_aircraft;
                Iterator<AircraftExt> it;
                int window, landing;
                Random rand = new Random();
```

```
                if(airListExt!=null){
                        for(int i=0; i<number; i++){
                                it = airListExt.iterator();
                                sol = new Solution();
                                while(it.hasNext()){
                                        temp = it.next();
                                        window = temp.getLatestLandingTime()-temp
                                             .getEarliestLandindgTime();
                                        if(window<=0)
                                                landing = temp.
                                                     getEarliestLandindgTime();
                                        else
                                                landing = temp.
                                                     getEarliestLandindgTime()+
                                                     rand.nextInt(window);

                                        new_aircraft = AircraftExt.
                                             createNewAircraftExt(temp);

                                        new_aircraft.setScheduledLandingTime(
                                             landing);
                                        sol.getListExt().add(new_aircraft);
                                }
                                sol.setFitness(eval.fintnessLinearExt(sol));
                                sol.setUnfitness(eval.unfitnessExt(sol));
                                //Finds the initial best solution
                                if(sol.getUnfitness()==0 && Math.abs(sol.
                                     getFitness())<Math.abs(bestSolution.
                                     getFitness()))
                                        bestSolution = sol;
                                population.add(sol);
                        }
                }
        }

        public ArrayList<Solution> getPopulation() {
                return population;
        }

        public Solution getBestSolution() {
                return bestSolution;
        }


}
```

A.4.2 Evaluation.java

```
package sorting;

import java.util.ArrayList;
import java.util.Iterator;

import model.Aircraft;
import model.AircraftExt;
import model.SeperationConstraints;
import model.Solution;

public class Evaluation {
        private SeperationConstraints sep;

        public Evaluation(){
                sep = new SeperationConstraints();
        }

        //Calculate the fitness, udsing a linear model-function
        public int fintnessLinear(Solution sol){
                int sum=0;
                Iterator<Aircraft> it = sol.getList().iterator();
                Aircraft air;

                while(it.hasNext()){
                        air = it.next();
                        sum += air.getTargetLandingTime()-air.
                             getScheduledLandingTime();
                }
                return sum;
        }

        //Calculate the fitness with target landing time as objective
        public int fintnessLinearExt(Solution sol){
                int sum=0, temp;
                Iterator<AircraftExt> it = sol.getListExt().iterator();
                AircraftExt air;

                while(it.hasNext()){
                        air = it.next();
                        temp = air.getScheduledLandingTime()-air.
                             getTargetLandingTime();
                        if(temp<0)
                                sum += Math.abs(temp*air.getPenB());
                        else
                                sum += temp*air.getPenA();
                }
                return sum;
        }

        //Calculate the fitness, udsing a squared model-function
        public int fintnessNonLinear(Solution sol){
                int sum=0;
                Iterator<Aircraft> it = sol.getList().iterator();
                Aircraft air;
                int temp;

                while(it.hasNext()){
                        air = it.next();
                        temp = air.getScheduledLandingTime()-air.
                             getTargetLandingTime();
                        if(temp>=0)
                                sum -= (temp*temp);
                        else
                                sum += (temp*temp);
                }
                return sum;
        }

        //Calculate the unfitness value
        public int unfitness(Solution sol){
                int sum = 0;
                int i = 0;
                int s,x2,x1;
                Aircraft air1, air2;
                ArrayList<Aircraft> list = sol.getList();

                while(i<list.size()-1){
                        air1 = list.get(i);
                        air2 = list.get(i+1);

                        s = sep.getSeperation(air1.getType(), air2.getType());
```

```
                        x1 = air1.getScheduledLandingTime();
                        x2 = air2.getScheduledLandingTime();

                        sum += Math.max(0, s-(x2-x1));
                        i++;
                }
                return sum;
        }

        public int unfitnessExt(Solution sol){
                int sum = 0;
                int i = 0;
                int s,x2,x1;
                AircraftExt air1, air2;
                ArrayList<Aircraft> list = sol.getList();

                while(i<list.size()-1){
                        air1 = (AircraftExt)list.get(i);
                        air2 = (AircraftExt)list.get(i+1);

                        s = air1.getSeperation(air2.getNumber());
                        x1 = air1.getScheduledLandingTime();
                        x2 = air2.getScheduledLandingTime();

                        sum += Math.max(0, s-(x2-x1));
                        i++;
                }
                return sum;
        }
}
```

A.4.3 Aircraft.java

```
package model;

public class Aircraft {
        private int number;
        private int apperanceTime;
        private int earliestLandindgTime;
        private int targetLandingTime;
        private int latestLandingTime;
        private int scheduledLandingTime;
        private int type;
        private boolean hasLanded;

        public Aircraft(int number, int elt, int tlt, int llt, int type){
                this.number = number;
                earliestLandindgTime = elt;
                targetLandingTime = tlt;
                latestLandingTime = llt;
                scheduledLandingTime = targetLandingTime;
                this.type = type;
                hasLanded = false;
                apperanceTime = -1;
        }

        public Aircraft(int number, int app, int elt, int tlt, int llt, int type)
            {
                this.number = number;
                apperanceTime = app;
                earliestLandindgTime = elt;
                targetLandingTime = tlt;
                latestLandingTime = llt;
                scheduledLandingTime = targetLandingTime;
                this.type = type;
        }

        public int getNumber() {
                return number;
        }

        public int getEarliestLandindgTime() {
                return earliestLandindgTime;
        }

        public void setEarliestLandindgTime(int earliestLandindgTime) {
                this.earliestLandindgTime = earliestLandindgTime;
        }

        public int getLatestLandingTime() {
                return latestLandingTime;
        }

        public void setTargetLandingTime(int targetLandingTime) {
                this.targetLandingTime = targetLandingTime;
        }

        public int getTargetLandingTime() {
                return targetLandingTime;
        }

        public int getType() {
                return type;
        }

        public int getScheduledLandingTime() {
                return scheduledLandingTime;
        }

        public void setScheduledLandingTime(int scheduledLandingTime) {
                this.scheduledLandingTime = scheduledLandingTime;
        }

        public static Aircraft createNewAircraft(Aircraft a){
                int number = a.getNumber();
                int elt = a.getEarliestLandindgTime();
                int tlt = a.getTargetLandingTime();
                int llt = a.getLatestLandingTime();
                int type = a.getType();

                Aircraft air = new Aircraft(number, elt, tlt, llt, type);

                return air;
        }
```

```
        public static Aircraft createNewAircraftAPP(Aircraft a){
                int number = a.getNumber();
                int app = a.getApperanceTime();
                int elt = a.getEarliestLandindgTime();
                int tlt = a.getTargetLandingTime();
                int llt = a.getLatestLandingTime();
                int type = a.getType();

                Aircraft air = new Aircraft(number, app,elt, tlt, llt, type);

                return air;
        }

        public String toString(){
                String temp= number + ":␣x:␣"+ scheduledLandingTime;
                temp += ":␣elt:␣" + earliestLandindgTime;
                //temp += " tlt: "+ targetLandingTime;
                //temp += " llt: "+ latestLandingTime;
                temp += "␣app:␣"+ apperanceTime +"\n";
                return temp;
        }

        public int getApperanceTime() {
                return apperanceTime;
        }

        public void setApperanceTime(int apperanceTime) {
                this.apperanceTime = apperanceTime;
        }

        public boolean isHasLanded() {
                return hasLanded;
        }

        public void setHasLanded(boolean hasLanded) {
                this.hasLanded = hasLanded;
        }

        public void setLatestLandingTime(int latestLandingTime) {
                this.latestLandingTime = latestLandingTime;
        }

}
```

A.4.4 AircraftExt.java

```
package model;

public class AircraftExt extends Aircraft{
        private double penB, penA;
        private int[] seperation;

        public AircraftExt(int number, int app, int elt, int tlt, int llt, double
              penB, double penA){
                 super(number, app, elt, tlt, llt, -1);
                 this.penB = penB;
                 this.penA = penA;

        }

        public void setPenA(double penA) {
                 this.penA = penA;
        }

        public void setPenB(double penB) {
                 this.penB = penB;
        }

        public double getPenA() {
                 return penA;
        }

        public double getPenB() {
                 return penB;
        }

        public int getSeperation(int i){
                 return seperation[i];
        }

        public int[] getSeperation() {
                 return seperation;
        }

        public void setSeperation(int[] seperation) {
                 this.seperation = seperation;
        }

        public static AircraftExt createNewAircraftExt(AircraftExt a){
                 int number = a.getNumber();
                 int app = a.getApperanceTime();
                 int elt = a.getEarliestLandindgTime();
                 int tlt = a.getTargetLandingTime();
                 int llt = a.getLatestLandingTime();
                 double penA = a.getPenA();
                 double penB = a.getPenB();

                 AircraftExt air = new AircraftExt(number, app,elt, tlt, llt, penB
                       , penA);
                 air.setSeperation(a.getSeperation());

                 return air;
        }

}
```

A.4.5 Solution.java

```
package model;

import java.util.ArrayList;
import java.util.Iterator;

public class Solution {
        private ArrayList<Aircraft> list;
        private ArrayList<AircraftExt> listExt;
        private int unfitness;
        private int fitness;

        public Solution(){
                unfitness = Integer.MAX_VALUE;
                //fitness = Integer.MIN_VALUE;
                fitness = Integer.MAX_VALUE;
                list = new ArrayList<Aircraft>();
                listExt = new ArrayList<AircraftExt>();
        }

        public void sort(String objective){
                //      Sort the aircraftList using bubble sort
                Boolean swapped = true;
                while(swapped==true){
                        swapped=false;
                        for(int i=0; i<listExt.size()-1;i++){
                                AircraftExt a = listExt.get(i);
                                AircraftExt b = listExt.get(i+1);
                                if(objective.equals("earliest")){
                                        if(a.getEarliestLandindgTime() > b.
                                             getEarliestLandindgTime()){
                                                listExt.set(i, b);
                                                listExt.set(i+1, a);
                                                swapped=true;
                                        }
                                }
                                else if(objective.equals("target")){
                                        if(a.getTargetLandingTime() > b.
                                             getTargetLandingTime()){
                                                listExt.set(i, b);
                                                listExt.set(i+1, a);
                                                swapped=true;
                                        }
                                }
                                else if(objective.equals("scheduled")){
                                        if(a.getScheduledLandingTime() > b.
                                             getScheduledLandingTime()){
                                                listExt.set(i, b);
                                                listExt.set(i+1, a);
                                                swapped=true;
                                        }
                                }
                                else if(objective.equals("apperance")){
                                        if(a.getApperanceTime() > b.
                                             getApperanceTime()){
                                                listExt.set(i, b);
                                                listExt.set(i+1, a);
                                                swapped=true;
                                        }
                                }
                                else if(objective.equals("number")){
                                        if(a.getNumber() > b.getNumber()){
                                                listExt.set(i, b);
                                                listExt.set(i+1, a);
                                                swapped=true;
                                        }
                                }
                        }
                }
        }

        public int getFitness() {
                return fitness;
        }

        public void setFitness(int fitness) {
                this.fitness = fitness;
        }

        public ArrayList<Aircraft> getList() {
                return list;
        }
```

```
        public void setList(ArrayList<Aircraft> list) {
                this.list = list;
        }

        public int getUnfitness() {
                return unfitness;
        }

        public void setUnfitness(int unfitness) {
                this.unfitness = unfitness;
        }

        public String toString(){
                String temp = "";

                Iterator<Aircraft> it = list.iterator();
                while(it.hasNext()){
                        temp += it.next().toString();
                }

                return temp;
        }

        public ArrayList<AircraftExt> getListExt() {
                return listExt;
        }

        public void setListExt(ArrayList<AircraftExt> listExt) {
                this.listExt = listExt;
        }
}
```

A.4.6 ImportAircrafts.java

```
package model;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.StringTokenizer;

public class ImportAircrafts {
        private ArrayList<Aircraft> aircraftList;

        public ImportAircrafts(String filename){
                aircraftList = extractAircrafts(readFile(filename));
        }

        public ArrayList readFile(String filename){
                ArrayList<String> incomingAircrafts = new ArrayList<String>();
                try {
                        FileReader fr = new FileReader(filename);
                        BufferedReader br = new BufferedReader(fr);
                        String temp = br.readLine();
                        while (temp != null){
                                incomingAircrafts.add(temp);
                                temp = br.readLine();
                        }
                        br.close();
                        fr.close();
                } catch (FileNotFoundException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
                return incomingAircrafts;
        }

        public ArrayList<Aircraft> extractAircrafts(ArrayList incomingAircrafts){
                aircraftList = new ArrayList<Aircraft>();
                int number=0, elt=0, tlt=0, llt=0, type=0, app=0;

                Iterator it = incomingAircrafts.iterator();
                while(it.hasNext()){
                        StringTokenizer temp = new StringTokenizer((String)it.
                            next());
                        Aircraft aircraft;
                        if (temp.countTokens()==4){
                                number++;
                                elt = new Integer(temp.nextToken()).intValue();
                                tlt = new Integer(temp.nextToken()).intValue();
                                llt = new Integer(temp.nextToken()).intValue();
                                type = new Integer(temp.nextToken()).intValue();
                                aircraft = new Aircraft(number, elt, tlt, llt,
                                     type);
                                aircraftList.add(aircraft);
                        }
                        else if (temp.countTokens()==5){
                                number++;
                                app = new Integer(temp.nextToken()).intValue();
                                elt = new Integer(temp.nextToken()).intValue();
                                tlt = new Integer(temp.nextToken()).intValue();
                                llt = new Integer(temp.nextToken()).intValue();
                                type = new Integer(temp.nextToken()).intValue();
                                aircraft = new Aircraft(number, app, elt, tlt,
                                     llt, type);
                                aircraftList.add(aircraft);
                        }

                }
                //Sort the aircraftList using bubble sort
                Boolean swapped = true;
                while(swapped==true){
                        swapped=false;
                        for(int i=0; i<aircraftList.size()-1;i++){
                                Aircraft a = aircraftList.get(i);
                                Aircraft b = aircraftList.get(i+1);
                                if(a.getEarliestLandindgTime() > b.
                                     getEarliestLandindgTime()){
                                        aircraftList.set(i, b);
```

```
                                        aircraftList.set(i+1, a);
                                        swapped=true;
                                }
                        }
                }

                return aircraftList;
        }

        public ArrayList<Aircraft> getAircraftList() {
                return aircraftList;
        }
}
```

A.4.7 ImportAircraftExt.java

```
package model;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.StringTokenizer;

public class ImportAircraftExt {
        private ArrayList<AircraftExt> aircraftList;
        private int freeze, planes;

        public ImportAircraftExt(String filename){
                aircraftList = extractAircrafts(readFile(filename));
        }

        public ArrayList readFile(String filename){
                ArrayList<String> incomingAircrafts = new ArrayList<String>();
                try {
                        FileReader fr = new FileReader(filename);
                        BufferedReader br = new BufferedReader(fr);
                        String temp = br.readLine();
                        while (temp != null){
                                incomingAircrafts.add(temp);
                                temp = br.readLine();
                        }
                        br.close();
                        fr.close();
                } catch (FileNotFoundException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
                return incomingAircrafts;
        }

        public ArrayList<AircraftExt> extractAircrafts(ArrayList
             incomingAircrafts){
                aircraftList = new ArrayList<AircraftExt>();
                int number=0, elt=0, tlt=0, llt=0, app=0;
                double penB, penA;

                StringTokenizer temp;
                freeze=0;
                planes=0;



                Iterator it = incomingAircrafts.iterator();
                if(it.hasNext()){
                        temp = new StringTokenizer((String)it.next());
                        planes = new Integer(temp.nextToken()).intValue();
                        freeze = new Integer(temp.nextToken()).intValue();
                }


                while(it.hasNext()){
                        temp = new StringTokenizer((String)it.next());
                        int[] seperation = new int[planes];
                        AircraftExt aircraft;

                        app = new Integer(temp.nextToken()).intValue();
                        elt = new Integer(temp.nextToken()).intValue();
                        tlt = new Integer(temp.nextToken()).intValue();
                        llt = new Integer(temp.nextToken()).intValue();
                        penB = new Double(temp.nextToken()).doubleValue();
                        penA = new Double(temp.nextToken()).doubleValue();
                        aircraft = new AircraftExt(number, app, elt, tlt, llt,
                             penB, penA);
                        number++;

                        //extract the seperation constraints
                        if(it.hasNext()){
                                temp = new StringTokenizer((String)it.next());
                                int i=0;
                                while(temp.hasMoreTokens()){
```

```
                                        seperation[i++] = new Integer(temp.
                                            nextToken()).intValue();
                                }
                                aircraft.setSeperation(seperation);
                        }
                        aircraftList.add(aircraft);
                }
                //Sort the aircraftList using bubble sort
                Boolean swapped = true;
                while(swapped==true){
                        swapped=false;
                        for(int i=0; i<aircraftList.size()-1;i++){
                                AircraftExt a = aircraftList.get(i);
                                AircraftExt b = aircraftList.get(i+1);
                                if(a.getEarliestLandindgTime() > b.
                                    getEarliestLandindgTime()){
                                        aircraftList.set(i, b);
                                        aircraftList.set(i+1, a);
                                        swapped=true;
                                }
                        }
                }

                return aircraftList;
        }

        public ArrayList<AircraftExt> getAircraftList() {
                return aircraftList;
        }

        public int getFreeze() {
                return freeze;
        }

        public int getPlanes() {
                return planes;
        }
}
```

A.4.8 SeperationConstraints.java

```
package model;

public class SeperationConstraints {
        private int[][] lookup= new int[5][5];

        public SeperationConstraints(){
                //Distances after a light or small aircraft
                for (int i=0;i<5;i++){
                        lookup[0][i]=3;
                        lookup[1][i]=3;
                }

                //Distances after a lower-medium aircraft
                lookup[2][0] = 6;
                lookup[2][1] = 5;
                lookup[2][2] = 3;
                lookup[2][3] = 3;
                lookup[2][4] = 3;

                //Distances after a upper-medium aircraft
                lookup[3][0] = 7;
                lookup[3][1] = 6;
                lookup[3][2] = 4;
                lookup[3][3] = 3;
                lookup[3][4] = 3;

                //Distances after a heavy aircraft
                lookup[4][0] = 8;
                lookup[4][1] = 7;
                lookup[4][2] = 6;
                lookup[4][3] = 5;
                lookup[4][4] = 4;
        }

        /**
         * getSeperation returns the seperation constrain in
         * seconds when a aircraft-type
         *  i is followed by aircraft-type j.
         *
         * @param aircraft type i
         * @param aircraft type j
         * @return seperation constraints in seconds
         */
        public int getSeperation(int i, int j){
                int temp = lookup[i-1][j-1];
                return temp*3600/160;
                //return temp*60/160;
        }
}
```

A.5 Final schedule after the dynamic tests

A.5.1 Approach 1

```
disp('Approach 1 - I')

disp('airland1.txt')
static = [98, 106, 121, 129, 137, 145, 160, 175, 184, 258];
dynamic = [100, 108, 121, 131, 139, 147, 162, 177, 185, 258];

disp('airland2.txt')
static = [92, 100, 109, 117, 125, 133, 141, 158, 173, 185, 197, 250, 313, 339,
     342];
dynamic = [95, 103, 111, 119, 127, 135, 143, 158, 173, 185, 197, 250, 313, 339,
     342];

disp('airland3.txt')
static = [82, 101, 109, 117, 131, 141, 150, 158, 166, 181, 189, 197, 229, 261,
     264, 287, 316, 336, 351, 433];
dynamic = [86, 106, 114, 122, 131, 141, 150, 158, 173, 188, 196, 211, 229, 261,
     264, 287, 316, 336, 351, 433];

disp('Approach 1 - II')

disp('airland1.txt')
dynamic = [100, 108, 121, 131, 139, 147, 162, 177, 185, 206];

disp('airland2.txt')
dynamic = [95, 100, 111, 119, 127, 135, 143, 158, 173, 181, 189, 204, 262, 339,
     342];

disp('airland3.txt')
dynamic = [86 106 109 117 125 133 141 149 164 179 187 202 205 212 225 248 261 272
      280 433];


disp('Approach 1 - III')

disp('airland1.txt')
dynamic = [100 108 121 131 139 147 162 177 185 206];

disp('airland2.txt')
dynamic = [95 103 111 119 127 135 143 158 173 181 189 204 262 277 287];

disp('airland3.txt')
dynamic = [86 106 114 122 130 138 146 154 169 184 192 207 210 213 225 248 261 272
      280 321];
```

A.5.2 Approach 2

```
disp('Approach 1 - I')

disp('airland1.txt')
static = [98, 106, 121, 129, 137, 145, 160, 175, 184, 258];
dynamic = [100, 108, 121, 131, 139, 147, 162, 177, 185, 258];

disp('airland2.txt')
static = [92, 100, 109, 117, 125, 133, 141, 158, 173, 185, 197, 250, 313, 339,
     342];
dynamic = [95, 103, 111, 119, 127, 135, 143, 158, 173, 185, 197, 250, 313, 339,
     342];

disp('airland3.txt')
static = [82, 101, 109, 117, 131, 141, 150, 158, 166, 181, 189, 197, 229, 261,
     264, 287, 316, 336, 351, 433];
dynamic = [86, 106, 114, 122, 131, 141, 150, 158, 173, 188, 196, 211, 229, 261,
     264, 287, 316, 336, 351, 433];

disp('Approach 1 - II')

disp('airland1.txt')
dynamic = [100, 108, 121, 131, 139, 147, 162, 177, 185, 206];

disp('airland2.txt')
dynamic = [95, 100, 111, 119, 127, 135, 143, 158, 173, 181, 189, 204, 262, 339,
     342];

disp('airland3.txt')
dynamic = [86 106 109 117 125 133 141 149 164 179 187 202 205 212 225 248 261 272
      280 433];


disp('Approach 1 - III')

disp('airland1.txt')
dynamic = [100 108 121 131 139 147 162 177 185 206];

disp('airland2.txt')
dynamic = [95 103 111 119 127 135 143 158 173 181 189 204 262 277 287];

disp('airland3.txt')
dynamic = [86 106 114 122 130 138 146 154 169 184 192 207 210 213 225 248 261 272
      280 321];
```

A.5.3 Approach 3

```
disp('Approach 1 - I')

disp('airland1.txt')
static = [98, 106, 121, 129, 137, 145, 160, 175, 184, 258];
dynamic = [100, 108, 121, 131, 139, 147, 162, 177, 185, 258];

disp('airland2.txt')
static = [92, 100, 109, 117, 125, 133, 141, 158, 173, 185, 197, 250, 313, 339,
     342];
dynamic = [95, 103, 111, 119, 127, 135, 143, 158, 173, 185, 197, 250, 313, 339,
     342];

disp('airland3.txt')
static = [82, 101, 109, 117, 131, 141, 150, 158, 166, 181, 189, 197, 229, 261,
     264, 287, 316, 336, 351, 433];
dynamic = [86, 106, 114, 122, 131, 141, 150, 158, 173, 188, 196, 211, 229, 261,
     264, 287, 316, 336, 351, 433];

disp('Approach 1 - II')

disp('airland1.txt')
dynamic = [100, 108, 121, 131, 139, 147, 162, 177, 185, 206];

disp('airland2.txt')
dynamic = [95, 100, 111, 119, 127, 135, 143, 158, 173, 181, 189, 204, 262, 339,
     342];

disp('airland3.txt')
dynamic = [86 106 109 117 125 133 141 149 164 179 187 202 205 212 225 248 261 272
      280 433];


disp('Approach 1 - III')

disp('airland1.txt')
dynamic = [100 108 121 131 139 147 162 177 185 206];

disp('airland2.txt')
dynamic = [95 103 111 119 127 135 143 158 173 181 189 204 262 277 287];

disp('airland3.txt')
dynamic = [86 106 114 122 130 138 146 154 169 184 192 207 210 213 225 248 261 272
      280 321];
```

A.5.4 Freeze

```
disp('freeze = 3, file = airland1.txt')
static = [98, 106, 121, 129, 137, 145, 160, 175, 184, 258];
dynamic = [103, 111, 124, 134, 142, 150, 165, 180, 188, 209];
me = mean(dynamic-static)
ma = max(dynamic-static)

disp('freeze = 5, file = airland1.txt')
static = [98, 106, 121, 129, 137, 145, 160, 175, 184, 258];
dynamic = [105, 113, 126, 136, 144, 152, 167, 182, 190, 211];
me = mean(dynamic-static)
ma = max(dynamic-static)

disp('freeze = 10, file = airland1.txt')
static = [98, 106, 121, 129, 137, 145, 160, 175, 184, 258];
dynamic = [110, 118, 131, 141, 149, 157, 172, 187, 195, 216];
me = mean(dynamic-static)
ma = max(dynamic-static)



disp('freeze = 3, file = airland2.txt')
static = [92, 100, 109, 117, 125, 133, 141, 158, 173, 185, 197, 250, 313, 339,
     342];
dynamic = [95, 100, 111, 119, 127, 135, 143, 158, 173, 181, 189, 204, 262, 277,
     287];
me = mean(dynamic-static)
ma = max(dynamic-static)

disp('freeze = 5, file = airland2.txt')
static = [92, 100, 109, 117, 125, 133, 141, 158, 173, 185, 197, 250, 313, 339,
     342];
dynamic = [95, 100, 111, 119, 127, 135, 143, 158, 173, 181, 189, 204, 262, 277,
     287];
me = mean(dynamic-static)
ma = max(dynamic-static)

disp('freeze = 10, file = airland2.txt')
static = [92, 100, 109, 117, 125, 133, 141, 158, 173, 185, 197, 250, 313, 339,
     342];
dynamic = [95, 100, 111, 119, 127, 135, 143, 158, 173, 181, 189, 204, 262, 277,
     287];
me = mean(dynamic-static)
ma = max(dynamic-static)



disp('freeze = 3, file = airland3.txt')
static = [82, 101, 109, 117, 131, 141, 150, 158, 166, 181, 189, 197, 229, 261,
     264, 287, 316, 336, 351, 433];
dynamic = [86, 106, 109, 117, 125, 133, 141, 149, 164, 179, 187, 202, 205, 212,
     225 248 261 272 280 321];
me = mean(dynamic-static)
ma = max(dynamic-static)

disp('freeze = 5, file = airland3.txt')
static = [82, 101, 109, 117, 131, 141, 150, 158, 166, 181, 189, 197, 229, 261,
     264, 287, 316, 336, 351, 433];
dynamic = [86, 106, 109, 117, 125, 133, 141, 149, 164, 179, 187, 202, 205, 212,
     225 248 261 272 280 321];
me = mean(dynamic-static)
ma = max(dynamic-static)

disp('freeze = 10, file = airland3.txt')
static = [82, 101, 109, 117, 131, 141, 150, 158, 166, 181, 189, 197, 229, 261,
     264, 287, 316, 336, 351, 433];
dynamic = [86, 106, 109, 117, 125, 133, 141, 149, 164, 179, 187, 202, 205, 212,
     225 248 261 272 280 321];
me = mean(dynamic-static)
ma = max(dynamic-static)
```