|  |
| --- |
| 01-02-2018 |
| Mokum Airways |
| How can we optimize one-day flight schedules? |

|  |
| --- |
| **Vrije Universiteit**  **Heuristics**  **BA2018**  Joyce Timmer - 2576122  Daphne Molenwijk - 2575089  Wouter den Duijn­­ -  Laura Holt – 2578129 |
|  |

# Introduction

Algorithms are a way of problem solving by following a specific set of rules. There are many different algorithms that work in all kinds of different ways. A constructive algorithm starts with an empty state and builds it up step by step, thus visiting all possible states. An iterative algorithm starts with a finished state and repeatedly modifies it in an attempt to find a better solution. Where a constructive algorithm might take longer, an iterative algorithm uses less memory. In order to quickly and efficiently solve a given problem, it is important to find (or create) the best algorithm for that problem.

In this report we will be examining the newly created airline Mokum Airways (MAW). We will create flight schedules in such a way that the airline will make the most profit. This is done by using different algorithms and comparing their results and efficiency. This report will look into which algorithm works best, thus produces the best results in the most efficient ways, for creating the optimal flight schedules for MAW.

Besides creating the flight schedules for the aircrafts, we also examine where MAW should have its home base, again by looking at the profits.

## Problem description

The airline Mokum Airways has six Airbus A321 aircrafts. Each aircraft flies at a speed of 800 km/h and has 199 seats available. MAW has landing rights at 28 destinations in Europe. Distances between all destinations are known. In the beginning, we consider Amsterdam the home base for MAW. By doing market research, MAW has checked how many people can potentially by transported at each of the 28 destinations. Every time an aircraft has to un-board and board passengers, it takes 1 hour in total.

It is given that the starting point of a route, thus the city in which an aircraft starts, has to be the same city as the final destination of the route, and aircrafts travel between 06:00 and 02:00. During a day, an aircraft has to land at least once at the home base for crew changes. Furthermore, the average speed of an aircraft equals 800 km/h and the maximal number of kilometres an aircraft can fly on one tank equals 3199 km. Aircrafts can refuel before the tank is empty and refuelling takes 1 hour.

MAW creates profit by flying passengers to their destinations. Passengers pay per travelled kilometre. Thus the distance between starting point of a passenger and his destination is the amount the passenger has to pay. It is possible to transport passengers to their destination via a detour. However, passengers still pay the same amount as flying directly to their destination.

The state space of the problem described is limited by a certain upper bound. When computing the upper bound of the state space we need to look at the maximum number of possibilities for adding cities and at all the different combinations of passengers that can be boarded. Given the fact that there are 20 hours in a day, the maximal number of cities in a schedule seems to be seven. For simplicity we take the number of cities in this calculation to be eight. When adding cities, for the first (and at the same time last) city we can choose a city out of all 28 possible cities. For the second until seventh city we can choose between 27 cities, since we do not choose the city we are currently in.

While determining all the different combinations of passengers in the route, we first looked at all the different booking possibilities of one seat. It turns out that there are 64 different combinations for a route containing seven flights if the seat is taken during all the flights. However, it is also possible that the seat is empty during some flights. Therefore, the number of different combinations for one seat equals 320.

This all leads to the following upper bound of the state space:

# Method

## Flight schedules

We use four different algorithms in order to try to find the optimal flight schedules: the random algorithm, the hill climber algorithm, the hill climber algorithm with restart and simulated annealing. All these algorithms are all iterative algorithms, except for the random algorithm. The random algorithm starts with an empty state, in our case flight schedule, and builds it up step by step. Using the randomly created flight schedule, the other iterative algorithms try and improve this finished state by certain mutations.

### Random Algorithm

The Random algorithm repeatedly generates random states, which are the flight schedules. The flight schedules are randomly created by starting at some random city, and randomly adding cities with random numbers of passengers.

We start by taking some random city as starting and also ending point. In this initial route the aircraft does not go anywhere. We know that we have to go to the home base (Amsterdam) once a day, so we check if the starting city is Amsterdam. If this is not the case, the first city we add to the route will be the home base city. After this check we continue with adding random cities between the starting and ending point at some random index (thus a random place in the route). When adding these cities it is checked whether it is possible to add this city, by looking at the distance between cities and the amount of fuel left in the tank. On any flight we put some random number of passengers, taking into account the potential passengers and the capacity of an aircraft. This is done by taking the minimum of the (remaining) capacity and the potential passengers for a flight, and taking a random number of passengers between zero and this minimum.

After generating one random schedule, a new one is generated. If the newly found schedule is better than the previous schedule, thus generates a higher profit, the new schedule is stored. If not, we forget the new schedule and keep the previous one. This is repeated a certain number of times, after which we will have the best schedule thus far.

### Hill climber Algorithm

The Hill climber algorithm starts with a random initial schedule and, through mutations, tries to improve this schedule. Possible mutations include removing or adding cities, adding passengers, swapping cities and swapping detours. When removing or adding cities we will again check if this is possible and reassign passengers. Since we start with a random schedule, it might occur that an aircraft is not full while there are still potential passengers left for that flight. For situations like these, it could be useful to add passengers to the flight. Swapping cities involves keeping all initial cities in the schedule but changing the order, and swapping detours means … .

Again, if a mutation improves the schedule, this better schedule is stored. If no improvement is found for a certain number of iterations, the flight schedule is fully optimized and returned. The hill climber algorithm does not guarantee to find the global optimum, the found optimized schedule might be in a local optimum.

### Hill climber Algorithm with restart

This algorithm starts the same as the normal hill climber algorithm. It starts with a random initial schedule and tries to improve it. In this case, however, after the initial schedule has shown to be as good as optimal (no more improvements), we start again with a new randomly generated schedule. Again we try to optimize this schedule with the hill climber algorithm. If it is found that this optimized schedule is better than the previous optimized schedule, we store the new schedule. This restart is done a certain number of times and returns the best optimized schedule. The chance of finding a global optimum increases when using the hill climber with restart instead of the ‘normal’ hill climber algorithm. This because we start again at another random place.

### Simulated annealing

Where the hill climber algorithm is likely to find a local optimum, simulated annealing is more likely to find the global optimum. In this algorithm we start with setting a high temperature and gradually letting it cool down. In the high temperature the algorithm accepts more solutions, thus also ones worse than the current best solution. By cooling down we let it accept fewer solutions, thus focussing on an area of the state space in which we hopefully find the optimal solution. By starting with a high temperature it allows the algorithm to possibly get out of a local optimum.

If the newly found solution is better than the previous solution we take the new solution and forget the old one. Otherwise, we accept a solution with a certain probability. This probability is given by the acceptance function and is defined as follows:

Difference means the difference between the new and the current profit. The temperature is initially set to some value and then cooled as we accept new solutions. It is cooled by some defined cooling rate. We will continue looking for a better solution until the temperature is equal to or lower than 1.

For the two parameters *Starting temperature* and *Cooling Rate* we will perform a grid search parameter tuning.

## Home base evaluation

In the previous situations, we took Amsterdam as home base. However, taking another city as home base might result in a higher profit. We will now examine whether Amsterdam is the best choice, given the profit.

We will look at the case of each of the 28 cities as home base and compare the best found flight schedules. The best schedules are found using the hill climber with restart.

# Results

Using the algorithms, we found different optimal solutions. Displayed are the timeline, map and profit in the optimal solutions for each situation and algorithm.

To equally compare different algorithms we need an equal number of iterations. In other words, we want an equal amount of evaluations per algorithm. An iteration is defined as a calculated state at some point in the algorithm. This can be a visited and a non-visited state.

For the results we ran each algorithm 50 times, where each run had an evaluation budget of 40,000 iterations. Thus in total 2 million iterations. In the table below we can see the time needed to compute for each of the four algorithms and the average and the highest profit generated for each algorithm. We can see that the hill climber algorithm runs the fastest, the best profit is generated by the simulated annealing and the best average profit by the hill climber algorithm with restart. Also, the random algorithm runs the slowest and produces the lowest profits.

Table 1: Results of 50 runs for each algorithm , where each run had 40,000 iterations, The runtime (in seconds) per run, the average and highest profit (in millions) are provided.

|  |  |  |  |
| --- | --- | --- | --- |
| Algorithm | RUNTIME | Best Profit | Average Profit |
| Random | 57.9 | 9.02 | 8.50 |
| Hill Climber | 6.66 | 11.27 | 10.64 |
| Hill Climber Restart | 22.18 | 11.54 | 11.19 |
| Simulated Annealing | 7.38 | 11.57 | 11.06 |

## Flight schedules

…

## Home base evaluation

…

# Discussion

Answer research question

Discuss results?

Discuss process?

Flaws/difficulties algorithm?

# References

…

# Appendix

Calculation upper bound?