Airline Scheduling with Simulated Annealing

Team CSB(3)
Chi Chun Wan – 2525244 Shabaz Sultan – 2566703 Boudewijn Zwaal – 1897527

***Abstract*—Abstract text to go here.**

I. Introduction

Flight schedules take up a central place in airline businesses. To maximize profit a business has to be as efficient as possible in using its resources and be as effective as possible in meeting marketplace demand. A business wants to plan its flights based on demand, use its airplanes to service these flights as efficiently as possible and assign crews to these flights to minimize expenses. These pose highly complex combinatorial optimization problems that cannot be solved analytically. Instead numerical optimization systems are used and their effectiveness can be influential on the profitability of a passenger focusses airline as a whole.
The Dutch airline KLM flies on 150 destinations with 97 airplanes and needs to produce a flight schedule four times a year that maximizes their profitability [1]. In this paper we will consider this problem in a slightly reduced scenario, with a fictional Mokum Airways that flies on 28 destinations with 6 airplanes. The goal is to maximize profit as measured by revenue passenger kilometers, often used as unit to measure the product being sold in a passenger focussed airline [2]. In order to obtain the best flight schedule, we use heuristics algorithms to analyze all the possible tours and find the optimal one.

II. Background

Constructing airline schedules can be broken down in a number of subproblems, each posing a number of computational and algorithmic challenges. Often first the flights that are provided by an airline are constructed, usually subject to complex regulation and non-deterministic customer demand functions. This is referred to as the Airline Scheduling problem in the literature, where the goal is to maximize profits by meeting said demands [3].
The next step is to assign the flights that are flown to airplanes. This is the so-called Fleet Scheduling Problem [4]. An airplane schedule can be constructed ahead of time, but scheduling methods for real world scenarios need to deal with real-time adjustments as well due to delays on the day itself (e.g. due to weather conditions, technical problems or issues at an airport). Ideally these schedules are repeatable (usually over either a one or seven day period), requiring that if at the start of the day an aircraft is at airport A, there needs to be an aircraft (not necessarily the exact same, just one of the same type) at airport A at the end of the schedule.
Finally crew personnel is assigned to the flights and airplanes. The crew scheduling problem is subject to constraints from flight authority regulation, labour regulation and airline specific regulation and because it represents the largest expenditure for airlines after fuel cost it is an important area of research for the industry as well [5].
This paper tackles the design and optimization of an airline schedule, with airplane assignment being part of the designed schedule. The one consideration given to the constraints from the crew schedule problem is that the schedule needs to have single homebase, each crew member is assumed to live at said homebase and each plane needs to attend that homebase daily for a crew change.
There are various approaches to this problem, such as integer programming [6] or even agent based approaches [7]. One candidate for complex combinatorial optimization problems in general is Simulated Annealing [8]. This meta-heuristic has been applied to related problems such as crew scheduling [9]. It has been specifically used for airline scheduling by [10], which is the main reference for the research described in this paper.

III. Problem

We consider an airline with a number of potential destinations and airplanes. The specific dataset used has 28 destinations and potential flights in both directions between every possible pair of destinations. We will analyse the problem with one and six airplanes. We wish to schedule a number of flights over a one day period. Based on the number of passengers and length of the flights a passenger kilometer score can be calculated for each of these flights. The total passenger kilometer score is the metric that needs to be optimized for. The number of passengers are based on a deterministic demand function, where for each potential flight the total passenger demand for the day is provided.
The flights are subject to a number of restrictions. In reality there are often regulatory flight restrictions only allowing take-offs and landing at certain hours of the day. We assume a 20 hour period of possible flight and that this 20 hour window is the same for every possible airport. This is a simplification that allows us to ignore timezone considerations. During this period there are a number of things that take up time. The flights will have a variable length of time, based on distance between start and end destination and speed of the aircraft (800 km/h in our example scenario). Docking takes up a certain amount of time (one hour in our example).
Beside time the planes' operation is also restricted by fuel consumption. The plane has a certain tank capacity that allows it a certain range before it has to refuel (3199 km in the example). If a plane wants to fly a route without having enough fuel in its tank it needs to refuel first. We will only consider the scenario where refueling will completely fill up the tank

and takes up a constant time (one hour).
Finally we have two airline requirements. There is the requirement to have the schedule be repeatable. To fulfil this requirement each plane will have its starting airport also be its final destination of the day. The airline also has a certain destination airport designated as homebase, which every plane needs to attend during the day for a crew changeover.

IV. Optimisation Algorithms

The most naïve way to approach an optimisation problem is to explore the entire state space and find the optimal solution in said space. We have developed a bruteforce algorithm to do just that, to be used for the simpler one plane scenario where bruteforcing is still computationally feasible. This allows us to apply simulated annealing to the same scenario and quantify its accuracy. We also developed a hill climber based on the same path generation and permutation code designed for the simulated annealer. The latter two algorithms can then be applied the more complex six plane scenario, where it is not computationally feasible anymore to bruteforce. The bruteforcer can be applied greedily to one plane at a time in the six plane scenario to allow for the same performance characteristics as the one plane scenario, but there is a danger of it finding a local optimum. The bruteforcer is however guaranteed to find the global optimum in the one plane scenario.

A. *Brute-force*

The state space of a plane's flight schedule consists of all possible valid tours. The way the realised brute force algorithm explores this space is by traversing a tree of potentially valid schedules using a depth-first search. This tree is illustrated in figure 1, with the homebase as the the root. Each node in the tree represents a potentially valid tour, each arrow going down being a flight and an additional flight from the node back the root being represented by a dashed grey arrow in figure 1. The depth of the tree is determined by a doing a preliminary check on the validity of each node's schedule. To do this the minimum time taken up by the node's schedule is calculated. We can calculate the time in the air for all the flights represented in the schedule. And if there are n flights, then there are $n-1$ dockings taking up time (the final docking can be outside our time window). And to calculate the time spent refueling the total flight distance is divided by the tank capacity distance (3199 km in our example scenario) and multiplied by the time taken for one refueling (60 minutes). This time can then be checked against the time window a schedule has. This test purposely underestimates the time for the potential schedules. Passing this test does however present a necessary, though not sufficient, condition for being an actually valid schedule. Thus it allows for a tree to be build, where a branch is terminated when it fails this preliminary test. During this preliminary test the flight back to the homebase is discounted, otherwise subtrees are cut that still contain valid schedules, even if the subtree's root node is not valid.

Attentive readers may have noticed that no consideration is

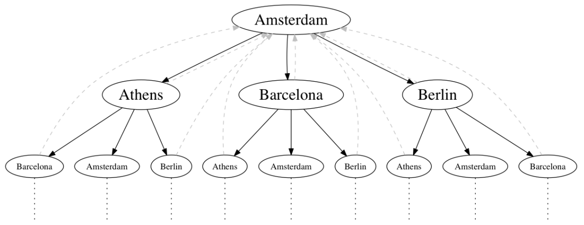

Fig. 1. Part of a tree representing potentially valid flight schedules in a system with only four cities, with Amsterdam as a homebase.

given to the concept of a starting airport of a schedule during tree construction. During the depth-first tree traversal every node visited goes through the first preliminary check. If it passes that check, the actual time spent refueling is calculated. This is done by trying all cities in a potential schedule as starting city and finding which choice of starting city would require the minimum number of refuelings. If the actual refuel time still leads to a total time that fits in the time window the schedule is considered valid. The score is then calculated. If the score is higher than the previous best encountered during tree traversal it is remembered. This will find the global optimum after the entire tree is traversed.
While this will works it is still fairly computationally intensive (wallclock time of a couple of hours using a C++ application). To shrink the tree more we can involve the best score found so far during traversal and use it to cut off subtree guaranteed to never produce better scores (i.e. a branch and bound algorithm). For each node a score can be calculated for all flights leading up to said not (not counting the flight back to the tree's root). All these flights will be part of all schedules in the node's subtree. We can also calculate a maximum score still achievable in the subtree of a node. To do this time spend so far getting from the root to the subtree is calculated (using essentially the same calculations as the preliminary schedule validity check). This gives a time left for the subtree. By assuming all the time left is spend in the air, with a plane full of passengers a maximum score that still can be achieved is calculated. If this score plus the score so far is less than the best score found so far the subtree is cut off. (Wallclock time goes down to under a second.)

B. *Hill-climbing*

Hill-climbing and simulated annealing share a lot of similarities. They both start with a random solution in state space and iterate on this solution using some permutation method. The permutation method tends to have a stochastic component. The hill-climbing algorithm at each iteration takes the current solution, permutes it and for both solutions calculates a score based on whatever metric is being used for optimisation. It then takes the solution with the best score as input for the next iteration. A schematic overview of the algorithm is presented in figure 2.

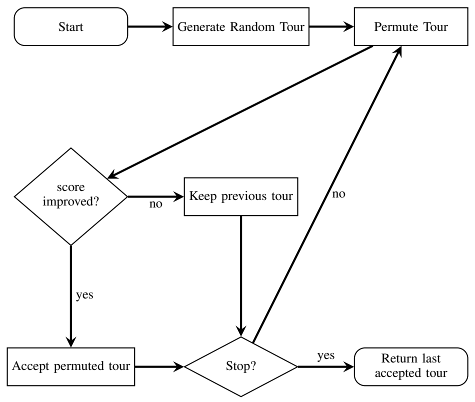

Fig. 2. Flowchart Hill-climbing

1) Tour generation and permutation: To make hill-climbing (and later simulated annealing) work, we need a way to generate tours and to permutate these tours. The way we generate a new tour is as follows. We start in our home-base, and then we add (randomly chosen) cities to the tour until the time is full. Then from the last chosen city we generate a tree-like structure that determines all possible routes from there. If one of those routes lead us back to our home-base within the time window, we accept that one. If such a route does not exist, we cut off the last city of the generated tour and try to reach our home-base again, with the same tree-like structure. If that is not possible again, we cut off the last city one last time and try again. If it is still not possible we try it again with a whole new route. After this we check what would be the most efficient city to start (and end) in, by checking how many times we need to refuel.
When permutating a tour, notice that we first shift it such that our home-base is again the first city in the tour. Then we cut off a random amount of cities from the tour (except the first one, which is our home-base). We then try to create a new tour from the two endpoints created in the route. We use the same algorithm for this as we did for the generated tour, i.e. we make a tree of all possible routes from one endpoint and check if one of them leads to the other endpoint. If not we cut off a city and try it again, just as when we generate a random tour. We then again check what is the most efficient start-end city.
This tour permutation method is inspired by [10]. They however after cutting a part the tour use a set of pre-generated trees that give all possible paths between two tours. They then pick one at random. This is a useful way to ensure that all options to fill up a gap in a tour are possible and that they are all equally likely to be chosen. Pre-generating these trees is however unfeasible for the number of airports we have. The proposed method is a way of hopefully preserving the desired properties of the full tree method, but without having to pre-generate unfeasibly large trees.

2) Hill-climbing: We start with an initial random tour generated with the algorithm described above. To find a better solution, we permutate the tour. The score of the passenger-kilometers for the initial tour is stored in a variable best score. If the score of the permutation tour is higher than the score of the arbitrary tour, the score of the permutated tour will be the new best score and the permutation tour will be accepted, otherwise the best score will stay the same and we will stay at the previous tour. This process is repeated until no further improvements can be found and the best tour is the last accepted tour. When applied to multiple planes, first one of the planes in the system is randomly chosen and its flightplan (tour) is permutated. The total score of all airplane tours is then calculated and compared to the previous best score. If it improves on said score, the new tour is accepted.
A disadvantage of hill-climbing is that it is good for finding a local optimum, but it is not guaranteed to find the best possible solution (global optimum). To avoid this disadvantage, we use a different algorithm, simulated annealing.

C. Simulated Annealing

Simulated annealing has the same iterative algorithm properties as the described hill-climber, using the same permutation method during iteration. And like hillclimbing, if a new tour is generated with a better score, it will always be used as input for the next iteration. However, when a permutation is encountered that does not improve the score, there is a chance it is used as input for the next iteration anyway. This allows the simulated annealer to more fully explore state space. The chance of it accepting a worse score is gradually lowered, so that it slowly transitions to a hill-climber, but ideally with a lower chance of it getting stuck in a local optimum than a straight up hill-climber. In figure 3, the flowchart of hill-climbing is adjusted in simulated annealing.

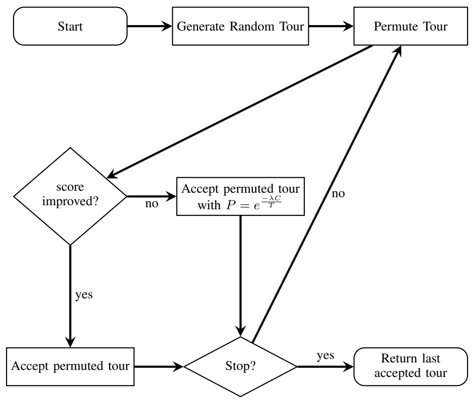

Fig. 3. Flowchart Simulated Annealing

In the case when the passengers-kilometer score is not improved, you will accept the permutation tour with probability

$$P = e^{\frac{\Delta C}{T}}, \tag{1}$$

where the difference in score is

$$\Delta C = C_{\text{permuted}} - C_{\text{last accepted}}. \tag{2}$$

The temperature T is gradually lowered as the iterations i go up, in a logarithmic manner using

$$T = \lambda^i \times T_0, \tag{3}$$

with λ being the cooling rate (values like 0.999, 0.9999, etc. used in our test runs) and T_0 the starting temperature. This probability will decrease with the number of iterations. Thus when the number of iterations increases, the temperature goes down. When temperature is high, the system will choose new states more or less at random, but as the temperature lowers this algorithm will go to hill-climbing.

V. Results

The first test scenario involves scheduling one airplane. The first simulations are run to observe general behaviour of the system. In figure 4 the results are plotted for a simulation with initial temperature $T_0 = 100\,000$ and cooling rate $\lambda = 0.9999$. We can see that the simulation moves from fairly random to a hill-climber as the temperature lowers. This matches the expected behaviour for simulated annealing and serves as an important sanity check. The second thing to observe is that with a start temperature of $100\,000$ the system goes through a significant period of randomness. Ideally we would like to pick a start temperature where the system doesn't have too much randomness and is starting to noticeably transition in its behaviour.
There are rigorous methods to do this, where the exact temperature is calculated that leads to a 50% acceptance rate. We have opted to judge things by eye and picked $T_0 = 50\,000$ as our start temperature. For six planes the start temperature is $T_0 = 50\,000 \times 6$. The score goes up roughly by six when going to six plane simulations and multiplying the temperature by six roughly preserves the ratio in the exponent of equation 1. The simulations are terminated when there is no change in the accepted schedule for 1000 iterations.

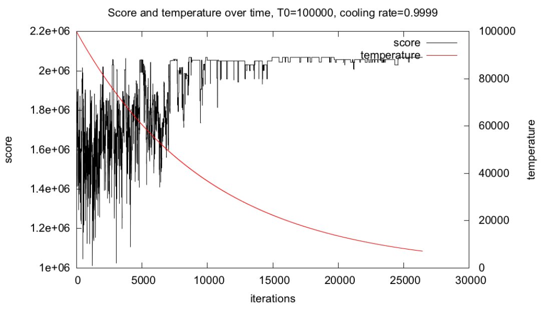

Fig. 4. Score over time for Simulated Annealing

A. *Best schedule for one plane*

For scenarios with one plane it is feasible to use a bruteforcer, as described in section IV-A. When looking at schedules that have Amsterdam as the homebase, the flightschedule in figure 5 is found, with a score of $2\,069\,202$. Finding optimal schedules for different homebases can be done by running the

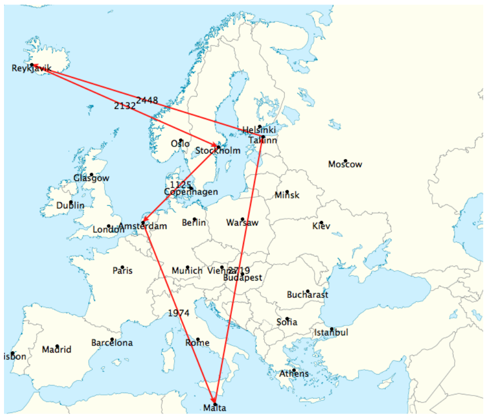

Fig. 5. Best tour for one plane with homebase Amsterdam, passenger kilometer score $2\,069\,202$.

bruteforcer with different cities as the homebase. The resulting scores are plotted in figure 6. We can see that the best schedule seems to include Barcelona, Helsinki and Reykjavik. This tour is plotted in figure 7, with a score of $2\,195\,766$. Any of the three cities can be chosen as start city.

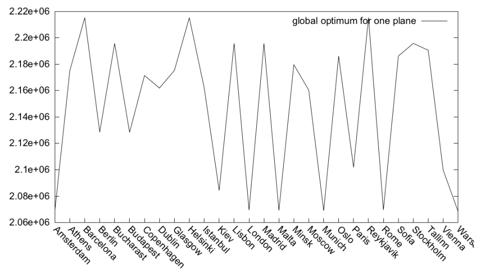

Fig. 6. Global optimum for different homebases with one plane

To analyze the accuracy of simulated annealing we can use

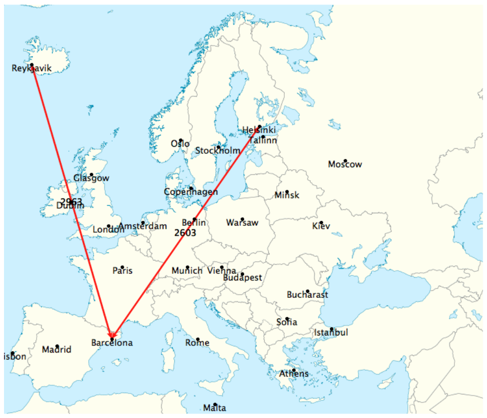

Fig. 7. Best tour for one plane with any homebase, passenger kilometer score $2\,215\,268$.

the global optimum as found by bruteforce for one plane with Amsterdam as homebase as reference. We run simulated annealing on the same scenario with a particular cooling rate a hundred times. The results of these testruns are presented in table I. The first thing to notice is that simulated annealing is able to find the global optimum and that it is increasingly likely to find if the cooling rate is slower. This serves as a further sanity check. We also observe that simulation time goes up if the cooling rate is lower. An iteration takes about 4-5ms on our test setup, which means that we move from a run taking roughly a couple seconds for $\lambda = 0.99$ to a run taking around 5 minutes with cooling rate $\lambda = 0.99999$.

Cooling Rate	Percentage global optimum found	Avg. nr. of iterations(std.dev.)
0.99	4%	1628.9($\pm$481.9)
0.999	6%	2804.3($\pm$1004.3)
0.9999	38%	13559.6($\pm$5149.9)
0.99999	83%	70111.5($\pm$20884.3)

TABLE I. PERCENTAGE OF 100 SIMULATED ANNEALING RUNS FINDING GLOBAL OPTIMUM, $T_0 = 50\,000$

We can further quantify the accuracy of the heuristic by calculating the root-mean-square error (RMSE) of the simulated annealing runs. This is presented as the black line in figure 8. We know that simulated annealing can find the global optimum and that if you invest more time by cooling more slowly the probability of finding that optimum go up and the RMSE goes down. We would like to know how effective simulated annealing is in spending its time compared to hill climbing. To this end a hillclimber is run for the same number of iterations as the average iteration counts of various simulated annealing cooling rate. This is presented as the red line in figure 8. For each iteration count a hundred runs have been executed. This hill-climber does not have a stop condition (i.e. quiting if the accepted schedule does not change for 1000 iterations). As an alternative the hill-climber is also run with this stop condition turned out, which means it will stop and restart multiple times if the iteration count is high enough. This is represented by the blue line, with again a 100 runs per iteration count.
We can see that for the same time investment, simulated annealing tends to have a lower RMSE than both hill-climber plots. This would point at simulated annealing making more effective use of the same time. The simulated annealer is better at avoiding local maxima than even running multiple hill-climbing runs for the same time investment.

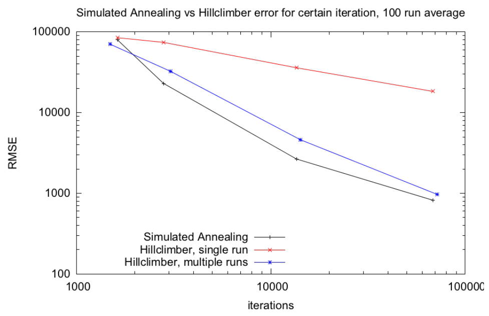

Fig. 8. Error for Simulated Annealing and Hill-climbing.

B. Best schedules for six planes

The result for the main problem with six airplanes is shown in figure(10) for different home-bases. The results are shown both for the brute-force algorithm as for the simulated annealing. Moreover, the score with six identical airplanes that fly the best route for one airplane is given to make an analysis of the algorithms. See Analysis section.
The best score with simulated annealing is 12 388 745 with as home-base Amsterdam. The best home-base to choose is Lisbon with the highest score of 12 908 931. Acquiring these results now took about 10-30 minutes. This has a large variance, because of the random nature of simulated annealing. What can clearly be seen is that most of the times the scores of the simulated annealer is not far off from our bruteforce approach. This indicates that our simulated annealer works quite well, further analysed in the analysis section.

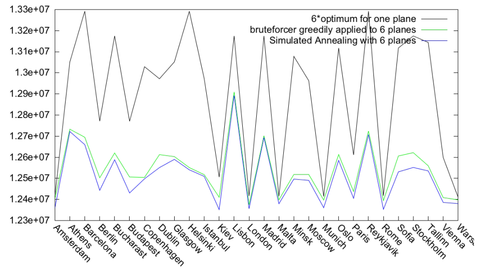

Fig. 9. Results for different homebases with six planes, Simulated Annealing line based on best of 5 runs for each scenario, cooling rate of 0.99999.

To evaluate our simulated annealing, the average number of iterations to find the optimum for the simulated annealing is given in the table below.

```
Average number of iterations: 328075.885714
Variance of the n.o.i.: 2025145743.66
Standard deviation of the n.o.i.: 45001.6193448
```

The simulation needed quite a lot iterations before terminating. For this data, we have done five runs per home-base for 28 different home-bases, that's a total of 5*28 runs. What's remarkable is that both the algorithms have about the same peaks as the same peaks as the theoretical six-times-one-plane situation. This indicates a strong correlation between these two and thus let us estimate the behavior of six planes for the behavior of one plane. See Analysis section.

VI. ANALYSIS

An interesting analysis is whether the solution for one airplane can predict the solution for six airplanes. In figure (8), the score times 6 for one airplane follows almost the same pattern as the score of the simulated annealing for six airplanes for different home bases. We ran a correlation test. This resulted in a correlation coefficient of 0.820 for simulated annealing. Therefore, we can conclude there is a correlation between the prediction for the score of six airplanes when the score for one airplane is known.

VII. Discussion

In our tests simulated annealing either equals or underperforms in terms of score compared to the greedy bruteforcer. One known shortcoming of the greedy bruteforcer is that it can get stuck in a local optimum, where the best set of routes for e.g. six planes does not necessarily contain the best route for one plane. As more planes are added to the test scenario the chance the greedy application of a per-plane bruteforcer encounters a local optimum goes up. And the chance of the difference with the global optimum and the found solution being significant go up as well. In theory a simulated annealer (or other stochastic methods) should be better able to deal with this problem. It would be useful to research the behaviour of the greedy bruteforcer and the simulated annealer applied to scenarios with a lot more than six planes.

We speculate that this may be the reason that [10] tested only with a scenario containing a high number of aircraft (46) and a low number of airports (12). This maximizes the probability that other methods get stuck in local maxima. The low number of airports also allows [10] to go with a relatively simple route permutation algorithm, where all possible routes are enumerated between two points and one is chosen. As the number of airports go up the number of paths becomes infeasibly large. These assumption are not made explicit by [10], but our efforts in partially replicating their methods make us propose these may be unstated reasons for their validation scenario choice.

While we do not generate a full tree of all possible paths possible between two airports like [10] for our path permutation code, we do have tree generation be a part of our algorithm. In the current version these trees are generated on-the-fly during each path permutation. An obvious candidate for performance optimization is to at least pre-generate a tree with a smaller time limit as bounding function. This could then possibly be integrated in the code so that generating code can be replaced by lookup code; an obvious application of the memoization optimisation technique.

Both the hill-climber and the simulated annealer use a Markov chain to traverse the state space of possible schedules. The Hastings-Metropolis algorithm allows you to sample the space stochastically even when the underlying probability distribution is unknown [11]. To show that the Markov chain is able to generate all possible paths it is necessary to prove ergodicity; i.e. to prove that the Markov chain is both irreducible and aperiodic. And to make sure that the application of Hastings-Metropolis that is part of simulated annealing is valid it should be shown that every possible ‘neighbour’ state is equally likely to be generated when permuting any of the possible routes. Proving both would be necessary if we’re interested in showing theoretical correctness in further research.

Finally we have done some exploratory work on applying Genetic Algorithms to this problem. The main challenge is constructing a method of creating new paths with two or more paths as its parents, with some additional mutation. This is equivalent to the construction of a permutation method being the primary challenge of applying a simulated annealing algorithm to a problem. From our literature study there are papers suggesting this approach to the airline scheduling problem and papers applying it to related problems like crew scheduling (e.g. [12]). But there is no literature actually applying it to the airline scheduling problem, so this may be interesting area of unexplored research.

VIII. Conclusions

In conclusion we can say that our simulated annealer works very well. After analyzing the various results we see that compared to the bruteforcer our simulated annealer performs very close to it. Also with a runtime of between 10 and 30 minutes to find an optimal schedule for six airplanes, it is not all that slow. Of course one can do the simulation faster, but we showed that the accuracy of the results then will decline. We can conclude as well that the simulated annealer works better than the hill climber, as discussed in the analysis section. We also showed that the outcome of scheduling six airplanes can be predicted with the output of the simulation with one plane. We found a high correlation between the global optimum of one airplane and the global optimum of six airplanes, and this can be seen also in figure (9), as the peaks agree of both sources.

References

[1] F. Bian, E. Burke, S. Jain, and G. Kendall, “Solution Methodologies for generating robust Airline Schedules,” *European Study Group ...*, no. February, pp. 9–19, 2003. [Online]. Available: http://www.wiskgenoot.nl/swi/Proceedings/Leiden2003/proceedingsSWI2003.pdf#page=15

[2] M. Schefczyk, “Operational performance of airlines: an extension of traditional measurement paradigms,” *Strategic Management Journal*, vol. 14, no. 4, pp. 301–317, 1993. [Online]. Available: http://www.jstor.org/stable/2486797

[3] M. Etschmaier and D. Mathaisel, “Airline scheduling: An overview,” *Transportation Science*, vol. 19, no. 2, pp. 127–138, 1985.

[4] R. a. Rushmeier and S. a. Kontogiorgis, “Advances in the Optimization of Airline Fleet Assignment,” *Transportation Science*, vol. 31, no. 2, pp. 159–169, 1997.

[5] B. Gopalakrishnan and E. L. Johnson, “Airline crew scheduling: State-of-the-art,” *Annals of Operations Research*, vol. 140, no. 1991, pp. 305–337, 2005.

[6] S. Raff, “Routing and scheduling of vehicles and crews,” *Computers & Operations Research*, vol. 10, no. 2, pp. 63–211, 1983.

[7] J. Langerman and E. Ehlers, “Agent-based airline scheduling,” *Computers & industrial engineering*, vol. 33, pp. 849–852, 1997. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0360835297002647

[8] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi *et al.*, “Optimization by simmulated annealing,” *science*, vol. 220, no. 4598, pp. 671–680, 1983.

[9] D. Sosnowska and J. Rolim, “Fleet scheduling optimization: a simulated annealing approach,” *Practice and theory of automated timetabling III*, 2001. [Online]. Available: http://link.springer.com/chapter/10.1007/3-540-44629-X_14

[10] J. Mashford and B. Marksjö, “Airline base schedule optimisation by flight network annealing,” *Annals of Operations Research*, pp. 293–313, 2001. [Online]. Available: http://link.springer.com/article/10.1023/A:1016027516013

[11] W. K. Hasting, “Monte Carlo sampling methods using Markov chains and their applications,” *Biometrica*, vol. 57, no. 1, p. 97, 1970.

[12] D. Levine, “Application of a hybrid genetic algorithm to airline crew scheduling,” *Computers & Operations Research*, vol. 23, no. 1, pp. 547–558, 1996.