Technische Universität Berlin

Institut für Mathematik
Fachgebiet Software und Algorithmen für die diskrete Optimierung

Fakultät II
Straße des 17. Juni 136
10623 Berlin

Seminar: Computational Integer Programming
Documentation

Airfreight Scheduling Problem

Wilhelm Bender
Johannes Jung
Konstantin Ntokas

02.07.2018

Supervised by
Prof. Dr. Thorsten Koch

Assistant Supervisor
Jakob Witzig

1 Introduction

A scheduling problem for airfreight is considered which takes a list of task flights as an input and wants to get a perfect schedule for each aircraft with respect to minimize the empty transportation. Hereby, the monitoring of the flight hours till maintenance should be tracked and accordingly the aircraft should be sent back to locations where maintenance can be operated. Additionally, for loading the freight into the aircraft certain loading systems are needed which needs to be at the origin and destination of a flight.
The problem is based on the fleet assignment problem which is explained in [5]. A mathematical formulation is given in section 3. The total optimisation is done in two steps. The first one just considers the scheduling problem of the aircraft with integrated flight hour monitoring and without the allocation of the loading systems. In the second step, the completed flight schedule is used to allocate the loading systems accordingly.
This is implemented with a mixed integer programming formulation and and calculated with SCIP. In section 4, some considerations of the implementation and computational history is presented.

2 Literature Review

The fleet assignment problem is a part of the airlines overall scheduling processes. In this process, scheduling designing is the first step which sets um a flight schedule according to the demand. This is further used as an input for the fleet assignment which assigns the flight legs with certain departure times to a type of aircraft. In [4], the aircraft routing process assigns a certain aircraft of the fleet to a certain flight. Further consideration with respect to maintenance and crew scheduling are done in [3].
In 1989, Abara formulated in [2] the fleet assignment problem in the form such that each variable represents a feasible connection between two flight legs. A few years later, a new approach was done in [5] where each flight leg is considered by the decision variable. This reduced the number of the variables and, therefore, became the most popular model for this problem type.
In [1], some integrated fleet assignment problems are introduced. The version with the integrated flight schedule assumes that a flight needs to be operated in a time window. Therefor, according to the time discretisation, the flight is copied to each time step in this time interval. Only one flight of these copied flights needs to be operated. Another integrated version shows a combination of the fleet assignment problem with maintenance constraints. Hereby, flights with the origin and destination being a city were maintenance can happen and flight time according to the maintenance time are introduced into the system. These flights need to be operated by each aircraft in a certain cycle time. Therefore, the accumulated flight time of an aircraft is not considered.
In this paper, we do not constraint the flight to a certain departure time. The departure time of the flights are only restricted by a deadline as an upper bound. Further, a new variable will be introduced which allows to track the amount of flight time till

maintenance of an aircraft in each time step.

3 Mathematical Model

In the first step, we consider the scheduling problem of the aircraft with integrated flight hour monitoring. The set of of the cities is assigned by C, the set of the task flights is assigned by L_task whereas L contains all task flights and some fictive flights. These fictive flights built up a complete graph based on C as nodes since it is possible to fly from each city to another. Flights are assigned by their origin and destination (p, d) the set of the aircraft is denoted by F. T is the time discretisation of the considered time interval.
x_{fpdt} represents the decision variable which means that $x_{fpdt} = 1$, if the aircraft $f \in F$ flies from city $p \in C$ to city $d \in C$ with departure time $t \in T$, and $x_{fpdt} = 0$ else. A further variable $y_{fp(t+0.5)}$ is the ground arc variable which is equal to 1, if the aircraft $f \in F$ stays on the ground at the city $p \in C$ during the time interval $[t, t + 1]$, and zero else.
The variable z_{ft} is responsible for the monitoring of the flight time of an aircraft because after a certain amount of flight time an aircraft needs to return to a certain location for maintenance. z_{ft} shows the number of remaining flight time for the aircraft f till maintenance at time t.
The following parameters are needed for the mathematical formulation. w_f^{max} indicates the maximal capacity of aircraft f, $\tilde{z_f}$ the maximal flight time till maintenance of aircraft f, w_{pd} the weight of the commodity on flight $pd \in L$, τ_{od} the flight time of flight $pd \in L$, τ_{pd} the deadline of flight $pd \in L$, $b \in C$ the base, where the aircraft are maintained. The aim is to minimise the total empty transportation amount. Therefore, a cost function $c_{fpdt} = (w_f^{max} - w_{pd})\tau_{pd}$ is used which measurer the cost for the aircraft f to fly the flight leg (p, d) in the sense of empty transportation. Fictive flights are assigned as an empty flight with $w_{pd} = 0$ and should be avoided. Then, the mathematical model to the problem is defined as the following integer programming formulation:

$$\min \quad \sum_{f\in F}\sum_{t\in T}\sum_{pd\in L} c_{fpdt}x_{fpdt} \tag{1}$$

$$s.t. \quad \sum_{f\in F}\sum_{t\in T} x_{fpdt} = 1 \quad \forall pd \in L_{task} \tag{2}$$

$$\sum_{d\in C} x_{fdp(t-\tau_{pd})} + y_{fp(t-0.5)} - \sum_{d\in C} x_{fpdt} - y_{fp(t+0.5)} = 0 \quad \forall t \in T, p \in C, f \in F \tag{3}$$

$$y_{fb(t-0.5)} = 1 \quad \forall f \in F \tag{4}$$

$$w_f^{max} - w_{pd}x_{fpdt} \geq 0 \quad \forall f \in F, pd \in L_{task}, t \in T \tag{5}$$

$$x_{fpdt}t + \tau_{pd} \leq \tau_d \quad \forall f \in F, pd \in L_{task}, t \in T \tag{6}$$

$$y_{fp(t+0.5)} \in \{0, 1\} \quad \forall f \in F, p \in C, t \in T \tag{7}$$

$$x_{fpdt} \in \{0, 1\} \quad \forall f \in F, pd \in L, t \in T \tag{8}$$

$$z_{f0} = \tilde{z_f} \quad \forall f \in F \tag{9}$$

$$z_{ft} \geq 0 \quad \forall f \in F, t \in T \tag{10}$$

$$z_{ft} = z_{ft-1} - \sum_{\substack{pd \in L \\ d \neq b}} x_{fpd(t-\tau_{pd})} \tau_{pd} + (\tilde{z_f} - z_{ft-1}) \sum_{pb \in L} x_{fpb(t-\tau_{pb})} \quad \forall f \in F, t \in T \tag{11}$$

The constraints can be sorted in four groups.
The first group consists of the constraints (7) and (8) and restricts the variables x and y to zero and and since these are decision variables as described above. The second group considers the continuity of each aircraft (2) and assigns each aircraft to the base as a starting point (4). The third group describes constraints which are related to the task flights. Equation (2) assures that every task flight is operated exactly once. Equation (5) ensures that the capacity of an aircraft is not exceeded by a task and equation (6) ensures that the deadline of an task is not exceeded.
The forth group considers the monitoring of the flight time till maintenance. The base is considered as the only place where maintenance can be operated. Equation (9) assumes that z_{ft} is a non-negative integer. Equation (10) assigns each aircraft a full "tank" of flight time in the beginning of the considered time interval and (11) ensures that the "tank" cannot be less than empty. Equation (12) gives the update of the remaining flight time till maintenance in each time step. The first part means that the flight time of a flight which does not return to the base is subtracted in time t if the arrival time is equal to t. The second part describes the case when an aircraft returns to the base, because then the "tank" is refilled to the maximal flight time till maintenance of aircraft. It stays the same as in the time step before if nothing happens in t.
These constraints build the basic model for the problem. Several constraints could be added to capture the problem in a more realistic way but are left out because of clarity.

In the second step, the transportation of the loading systems is optimized such that the the loading systems are allocated according to the optimized flight schedule from the first step.
Therefore, the set of the loading systems is assigned by A and the parameter w_l gives the maximal possible lifting weight of the loading system l. The cost function is denoted by $c_{fodt} = 0.25 * w_l * \tilde{\tau_{pd}}$ as it is assumed that the own weight of the loading system is 25% of the maximal possible lifting weight. $\tilde{\tau_{pd}}$ describes the transportation time of a loading system from city p to city d. It is assumed that the transportation time is linearly related to the flight time.
As in the first step, the variable a_{lpdt} is the decision variable and $b_{lp(t+0.5)}$ is the ground arc variable. w_{pd} stays the weight if the transported commodity of the flight (p, d).

Then, the problem can be formulated as:

$$\min \quad \sum_{f\in F}\sum_{t\in T}\sum_{pd\in L} c_{fpdt}a_{fpdt} \tag{12}$$

$$s.t. \quad \sum_{d\in C} a_{fdp(t-\tau_{pd})} + b_{lp(t-0.5)} - \sum_{d\in C} a_{fpdt} - b_{lp(t+0.5)} = 0 \quad \forall l \in A, p \in C, t \in T \tag{13}$$

$$b_{l\tilde{p}(-0.5)} = 1 \quad \forall l \in A, \tilde{p} \in C \tag{14}$$

$$b_{lp(t+0.5)} \in \{0,1\} \quad \forall l \in A, p \in C, t \in T \tag{15}$$

$$a_{lpdt} \in \{0,1\} \quad \forall l \in A, pd \in L, t \in T \tag{16}$$

$$\sum_{\substack{l\in A \\ if\, w_l \geq w_{pd}}} b_{lp(t-0.5)} \geq 1 \quad \forall x_{fpdt} \text{ if } pd \in L_{task} \tag{17}$$

$$\sum_{\substack{l\in A \\ if\, w_l \geq w_{pd}}} b_{lp(t+\tau_{pd}-0.5)} \geq 1 \quad \forall x_{fpdt} \text{ if } pd \in L_{task}$$

The constraints $(14) - (17)$ are analogue to the constraints in the first step. The constraint (18) ensures that there is a loading system available at the origin at departure time and at the destination at arrival time of a flight.

4 Implementation

4.1 Introduction

Some further considerations about the implementation will be presented in this section. Before solving the optimisation problem, the required data is extracted from the input. As an input of the entire model, the number of possible cities and a list of task flights is required. The origin and destination of the task flights needs to be out of the set of possible cities. We assume that no task flight includes the base as an origin or destination. In the model a flight (p, d) is only dependent on the origin p and destination d. Therefore, some adjustments are made to ensure several flights with the same origin and destination. In this case, different layers are used to assign each flight uniquely. If two flights with same origin and destination are in the task list, the number of possible cities is added to the number of the origin and destination. Hence, one flight is pushed in the second layer. To illustrate it better, a little example is given. Assume the number of possible cities is 100 and two task flight are given from city 1 to 2. Then the first flight is assigned by $(1, 2)$ and the second flight by $(101, 102)$. This can be repeated several times. In the end, the fictive layer lies to top. It consists of a complete graph and also includes the base. Therefore, in the little example above, we obtain the flights $(201, 202), (202, 201), (0, 201), (201, 0), (0, 202), (202, 0)$ as set of fictive flights. This ensures that it is always possible to fly from each city to another. Joining both, the flight task and the fictive flight sets, gives the entire set of flights.

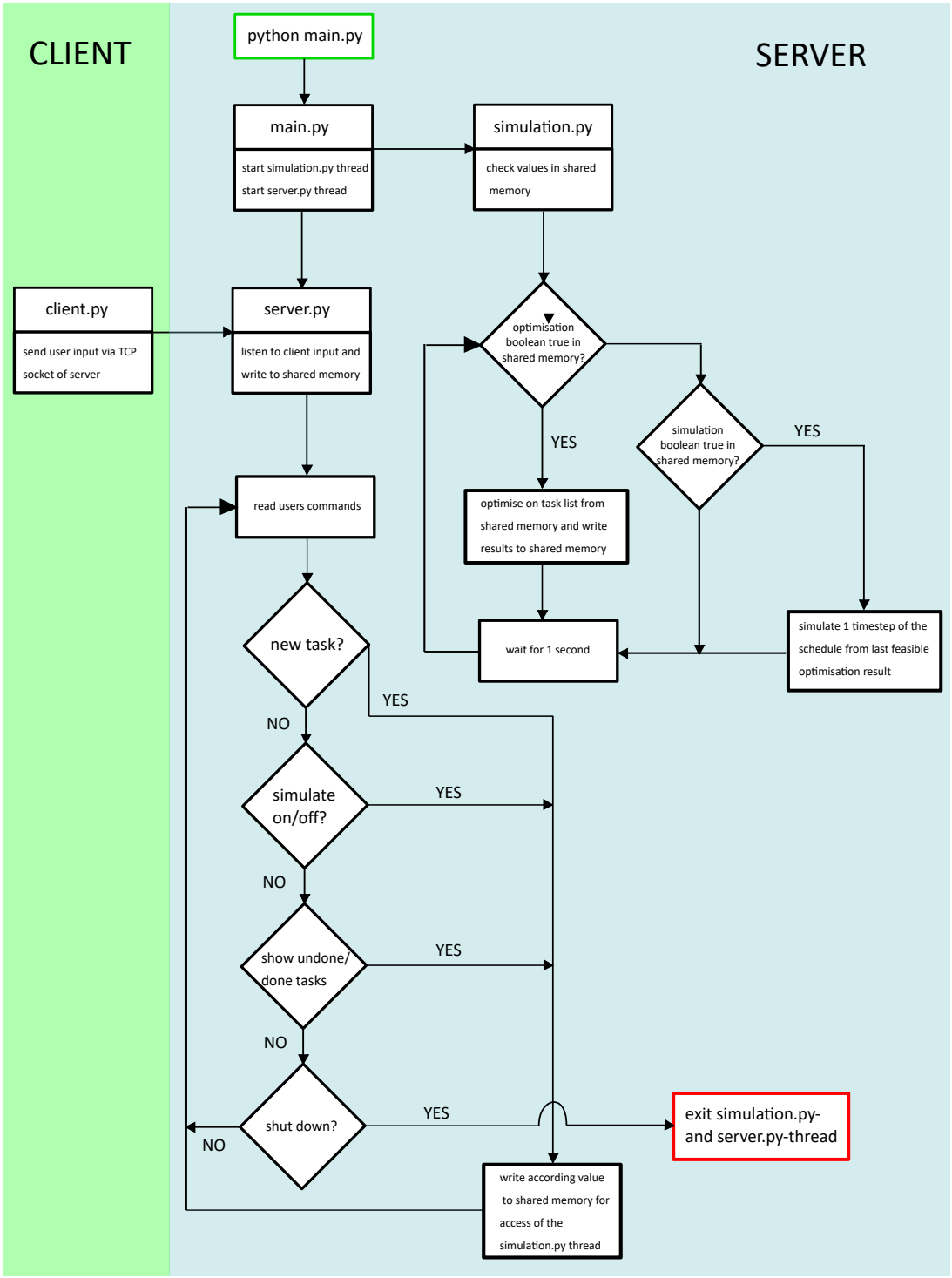
The set of the aircraft is given by:

number of aircraft	aircraft type	payload (in tonnes)
1	AN-225	250
7	AN 124-100	150
1	AN-22	60
1	AN-26	5.5

The set of the cities is extracted by the set of flights. In the little example, the set of the cities is given by $0, 1, 2, 101, 102, 201, 202$.
The discretised time interval is bounded by the maximum of the deadline of the task flights.
Furthermore, a flight time matrix is generated randomly. It is based on the number of possible cities and has the flight time from one city to another as entries. Also, a fixed loading time is added to the flight time.
These are the required sets for optimisation of the scheduling problem of the aircraft.
In the second set we use the flight schedule from the first step. Every aircraft is already equipped with a loading system with a payload of 30 tonnes. The number of the loading systems can be adjusted. Therefore, the feasibility can be tested with different number of loading systems. The loading system are distinguished by different payload. Then the routes of each loading system is optimised such that the transportation cost is minimised.
These sets are used to optimise the problem by using SCIP with the python interface PySCIPOpt. The implementation of the problems in the above sections is made publicly available on the website `https://github.com/globalw/ilp_schedule`. Hereby, we implemented a server which tracks the aircraft and allows to integrate a new task at a later point. This initialise a new optimisation of the new task and the tasks which have not been operated till that point. If the new optimisation problem is infeasible, the entire model stops. Therefore, in further work, we would like to remove the newly integrated task flight and continue with the operation of the previous optimisation.

4.2 Outline

We also want to give a rough overview on how the software is running with a perspective on the server and client environment. The Software includes a client application, that is communicating with a server via a TCP-socket. In general this software is able to be run on two different machines in a common network, such that the server side, which is doing the tough computational work, can be chosen a very large system with many computational resources, and the client may be chosen to be very weak. The server receives and understands commands such as to start an optimisation, integrate tasks, to shut down and to give orders to aircraft or vessels transporting cargo from one point to another. In the software the **simulation.py**-file implements a thread that runs in parallel to the server.

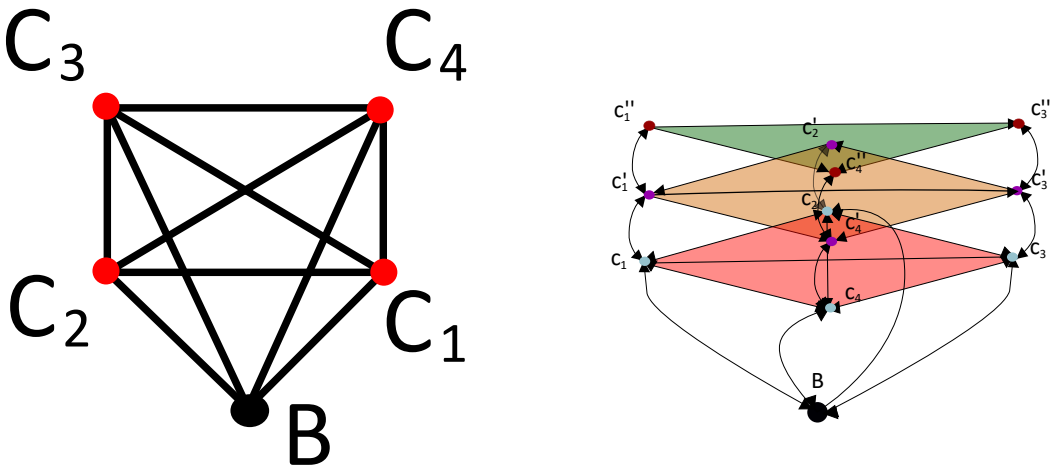


If a result of an optimization is available, the thread will await the command "simulate true" from the client and start going through the time dimension of the optimised de-

cision variable. The simulation of the schedule through out time is currently set to do one timestep every 0.5 seconds. For controlling a live system, this variable would need to be put to a more realistic value as for example one hour. So all in all this thread is able to simulate the schedule of the aircraft but also direct the actions of the aircraft in a life system.
Below you see a flowchart representation of the software with the description we just made.

4.3 Concept

So far we have not gone into the detail of the exact implementation of the model. We want to illustrate one of the most important concepts here, because it takes up a large portion of the semantics inside the optimization problem formulated with python. The general idea of how to organize the different tasks and access them later on in a smart way is the graph of the cities included in the task list.



Suppose we have the graph on the left as the fully connected graph representing the base B and cities C_1 to C_4. On the right we see a representation of the very same graph where we consider our current list of tasks. The layer in red is the same graph as on the left. Its weights are a triplets consisting of $(\delta, \zeta, \tau) \in \mathbb{R}^3$ with δ being the deadline time, ζ being the cargo to be transported and τ the time the aircraft needs for this flight. So basically this layer is used in order to enable aircraft to fly from one city to another without carrying any cargo.We call this layer the fictive layer and denote its adjacency matrix by $L_fictive$. The layers above represent task flights at all times and its adjacency matrix is denoted by L_task. So when there is a task route to be flown from C_1' to C_3' at some time $\tilde{t}$ and another task going in the same direction at some different time $\hat{t}$, then we will have a new layer with cities C_1'' to C_3'' such that this task is unique. With this abstraction we are able to make different cargo planes fly tasks even at the same time. The cargo plane is able to move between the layers without any cost because the time weights are defined to be zero.

If we remember the cost coefficient for the airfreight problem, it is calculated by

$$c_{fpdt} = (\omega_f^{max} - \omega_{pd})\tau_{pd} \tag{18}$$

where ω_f^{max} was the capacity of the aircraft f, ω_{pd} the cargo weight of the plane moved from city p to city d and τ_{pd} its time of flight. To traversing from layer to layer add no cost to the objective function value if the time is defined to be zero for the edges between the layers. Yet cost is added when a plane is flying to any city without any cargo loaded.
Inside the implementation we build a python dictionary of the following form:

$$\begin{aligned} L_{task_i} &: (\delta, \omega, \tau)_{o_i,d_i} \\ L_{fictve} &: (\infty, 0, \tau) \\ L_0^+ \text{ or } I^+ &: (\infty, 0, 0)_{p_i,d_{i+1}} \\ L_0^- \text{ or } I^- &: (\infty, 0, 0)_{p_i,d_{i-1}} \end{aligned}$$

L :

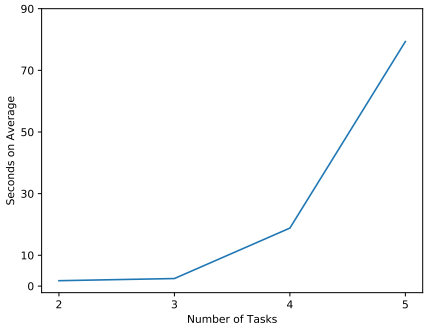
$$\begin{pmatrix} L_{task_1} & I^+ & 0 & 0 \\ I^- & \ddots & I^+ & 0 \\ 0 & I^- & L_{task_n} & I^+ \\ 0 & 0 & I^- & L_{fiktive} \end{pmatrix}$$

L represents the graph from the illustration above and is used in the implementation with much convenience. Additional ideas of for example to include loading systems can be applied to this graph model without interference with the code that has been written until now.

5 Computational Results

One of the bottle necks in a real time application is the amount of time between the end of a current flight up to a different route that is calculated on behalf of a new task list coming in. In the worst case the director of the flight schedule will have the time step by which the schedule is discretised. Suppose the schedule is discretised to one hour, then the optimizer will have less or equal to an hour before it can come up with a new schedule, that includes the newly received tasks. The longer the optimization takes, the higher the risk, that following the old schedule will result in a non-optimal or cost intensive schedule. We compute randomliy generated task lists for which we set up a deadline between the next ten to hundred hours. Then we linearly increase the number of tasks in the list and measure the time of ten such arbitraraly generated task lists. The closer one of the deadlines is to one hundred hours the more time the optimizer will need in order to find a solution. This is apart from the general runtime complexity of the problem an additional factor since the size of the time discretisation depends on the highest deadline time in the task list. The runtime measuement has been carried out on

a 2.70 GHz single core virtual machine. Each instance with its according task list size has been calculated twenty times and the computation times have been averaged.



Despite that we only have calculated up to a task size of five tasks, the computation took very long. This is due to the fact, that we have had tasks in the task list, that had dead line times of up to a hundred hours. Therefore the time to find a feasible solution can grow quickly because we have a decision variable with a dimension of up to a hundred timesteps. For Application one might think about making the time discretisation more coarse or chose a time grid with arbitrary timesteps.

6 Outlook

We turned the projects into a open source project on github because we would like to continue on this project and specialise the model to further realistic requirements. The biggest step would be to build a aircraft schedule model with integrated loading system allocation. This would enable that loading system and aircraft could wait for each other. It also would reduce the infeasibility of the model. Further work can be done by including loading times which depends on the weight of the task flight. Also, the model could be run with different time discretisations. Intuitively, the higher the resolution of time, the more optimal the result should be. Of course this causes a higher computational cost. Therefore the relation between computational cost and time discretisation would be very interesting.

References

[1] Airline fleet assignment concepts, models, and algorithms. *European Journal of Operational Research*, 172(1):1 – 30, 2006.

[2] Jeph Abara. Applying integer linear programming to the fleet assignment problem. *Interfaces*, 19(4):20–28, 1989.

[3] L. W. Clarke, C. A. Hane, E. L. Johnson, and G. L. Nemhauser. Maintenance and crew considerations in fleet assignment. *Transportation Science*, 30(3):249–260, 1996.

[4] Lloyd Clarke, Ellis Johnson, George Nemhauser, and Zhongxi Zhu. The aircraft rotation problem. *Annals of Operations Research*, 69(0):33–46, Jan 1997.

[5] Christopher A. Hane, Cynthia Barnhart, Ellis L. Johnson, Roy E. Marsten, George L. Nemhauser, and Gabriele Sigismondi. The fleet assignment problem: Solving a large-scale integer program. *Mathematical Programming*, 70(1):211–232, Oct 1995.