The Intelligent Decision of Flights Adjusting Rule-based flight scheduling optimisation

Sichen Zhao
School of Science, RMIT University,
Melbourne, VIC 3000, Australia
sichen.zhao@rmit.edu.au

Wei Shao*
School of Science, RMIT University,
Melbourne, VIC 3000, Australia
wei.shao@rmit.edu.au

Haitao Zhu
School of Science, RMIT University,
Melbourne, VIC 3000, Australia
eric.zhu@rmit.edu.au

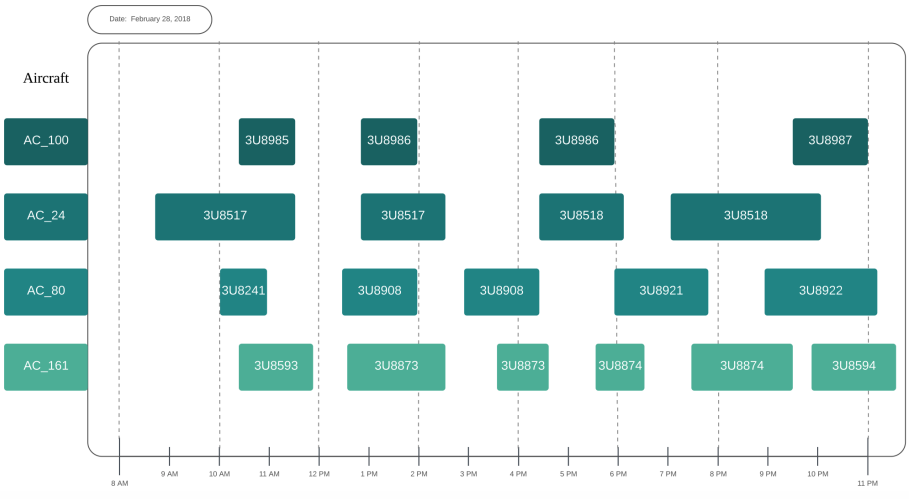

Figure 1: A time-line example for four selected aircraft

ABSTRACT

Aircraft plan scheduling is a significantly important problem to airport traffic controllers and managers. Airport traffic controllers need to arrange the order of each aircraft in order to save time and avoid collisions. There are a couple of options such as delay. switch, using ideal flight and cancel. We develop a rule-based searching strategy using optional actions in this paper to minimize the objective functions value. We also design and implement the proposed method with high effectiveness and efficiency. In this report, we will show the framework design and details of the proposed solution. The running results is also included.

KEYWORDS

Flight schedule, Uniform cost search, Disruptions management

WSDM '19, February 11–15, 2019, Melbourne, Australia

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

ACM Reference Format:
Sichen Zhao, Wei Shao*, and Haitao Zhu. 2019. The Intelligent Decision of Flights Adjusting Rule-based flight scheduling optimisation. In *WSDM '19, February 11–15, 2019, Melbourne, Australia.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In the recent years, data-driven solutions are increasing popular to many real-world problems. It has been widely used in smart cities and infrastructures [3]. Flight scheduling problem plays an crucial role in smart cities area. In this paper, we briefly introduce a relevant task and how we solve this problem.

The task is aiming to design a set of flight intelligent scheduling algorithms to solve the flight delay problem of Sichuan Airlines with airport capacity constraints and maintenance flexibility. A clear objective function and multiple restrictions are given. In order to obtain a solution that minimizes the objective function with strict constraints, this report defines the reschedule flights' problem as a typical optimization issue and builds a framework to generate a valid and efficient solution for this task.

The problem can be explained in three aspects. Firstly, a flight schedule validity evaluation is important in this task. All flight scheduling solutions should satisfy the limitations. In this report, a function is designed to check each generated solution during the process. If the solution does not fit the restrictions, the algorithm

will make other attempts. Secondly, how to find the best adjustment for each delayed flight is the key problem. In this report, there are five operating options. Four candidate operations are given by the task. Another operation is 'transfer', which is transferring the passengers to another plane. To select the most suitable operation for each delayed flight based on both the current flight order and the limitations, a rule-based strategy is created to find the reasonable operation for the delayed flight or conflicted solution. The conflicted solution is the result that unsatisfied with the restrictions. Thirdly, the optimization algorithm is crucial. Optimization algorithms usually are computationally expensive. Since the task has time limitations, shrinking the search space and adding some heuristic knowledge helps speed up the process. This report incrementally polishes the code and optimizes the data type to reduce the consumption time.

Since for most of the airline companies, the most valuable assets are aircraft [2] and given the fact that all those flights can be treated as continuous trajectories for each individual aircraft. It is crucial to solve the potential conflicts on each aircraft. And because the overall searching state space is overwhelmingly big for the whole 2200 flights, decomposing it to different aircraft-based plan effectively decrease the decision space and pruning is also used to increase the efficiency of our model.

In this report, the framework of the problem-solving process will be interpreted first. Then, the design of the class is illustrated. In the next section, we present the final results of the task. At last, a final conclusion will be drawn.

2 ALGORITHM

As introduced in the previous section, we decompose the whole flights plan into multiple aircraft-based plan. And as shown in the pseudo-code, after decomposing the original flights plan, the model will sort the conflicts based on a weighted importance factor, and then solve them one by one by using a extended Dijkstra algorithm [1].

To limit the space of the decision space and there is a very high penalty for a aircraft to change its base airport, we apply special rules on the 'idle' and 'switch' actions to make sure that the model will only use these actions when the aircraft has a 'loop' trajectory. An example is shown in the figure 2 It has a significant effect on increasing the efficiency but meanwhile, the methods lose it capability to find the global optimal solution.

The flowchart of our framework is shown in figure 3. First, we import the original plan from the dataset provided by Sichuan Airline. The original plan consists of the flight information, order of flights and timetable. We extract that information from .csv file and set the start state. Then we use a conflict check module to check the validity of this state. We will choose the corresponding action (method) to existing conflicts until all conflicts have been resolved. Finally, we add all chosen actions to the start state and change the state of flight schedule. The final result is the combination of the state and the action lists.

3 CLASS DESCRIPTION

In our model, we have five main classes, each of them consists of some specific information or functionality.

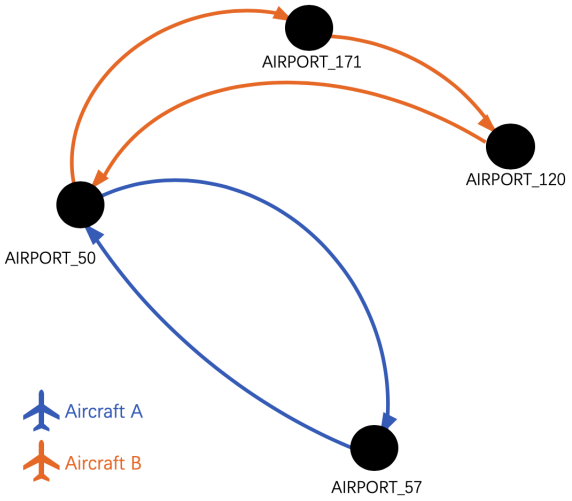

Figure 2: An example of loop route

Algorithm 1 Uniform Search with two-layered Priority Queue

Require: Start state
Ensure: Path to goal state

$conflicts \leftarrow$ new Priority Quere ordered by descending (Importance Factor)
for $conflict \in startState.conflictList$ **do**
  $conflicts.insert(conflict)$
end for

while not $conflicts.empty()$ **do**
  $currentConflict \leftarrow conflicts.pop()$
  $openList \leftarrow$ new Priority Quere ordered by ascending cost
  $open.insert(state(current))$
  $closed \leftarrow \phi$
  while not $open.empty()$ **do**
    $\sigma \leftarrow open.pop()$
    if $state(\sigma) \notin closed$ **then**
      $closed \leftarrow closed \cup state(\sigma)$
      if currentConflict is solved **then**
        break
      end if
      for each$(a, s') \in$ succ$(state(\sigma))$ **do**
        $\sigma' \leftarrow make - node(\sigma, a, s')$
        $open.insert(\sigma')$
      end for
    end if
  end while
end while

if not $state(\sigma).isgoal()$ **then**
  extract-solution
end if

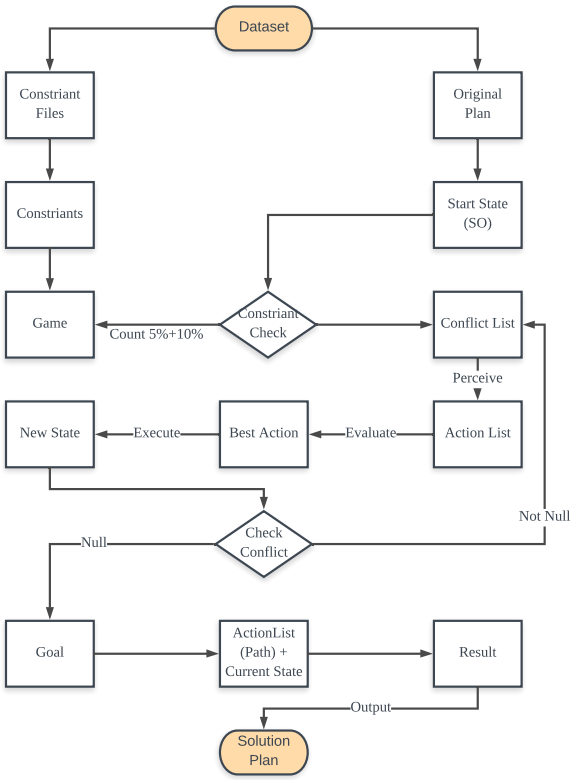

Figure 3: Flowchart

The first one is class "Game". It is the platform of this project and contains most of the basic information that needed, from hard constraint data to each plane object. The second one is class "State". It is the basic unit in this model, and each represents one arrangement of all flights in the plan. By applying different actions to a state, we will get different adjustment plans, and we can run the function "outputToCsv" to get the adjusted result.

Class "Flight", the sub-unit of the class "State", comprises all of the information for a certain flight.

Class "start", technically speaking, it isn't a class-type thing, but it contains the search algorithm that we used in this project and also can be used as main function in command line prompt.

The last one is class "util". Same as "start", it isn't a class-type thing, but it contains all of the support function, including computing object function and transfer factor, etc.

The brief description of other classes in our model is shown in the table below.

Class Name	Description
Airport	Comprise essential information for a certain airport (ID, permit aircraft type, standard turn around time, etc).
Aircraft	Comprise essential information for a certain aircraft (ID, type, number of seat, etc).
AircraftType	Enumerate type that include all of the aircraft types (type A-D).
Action	Comprise the type of a certain action and the details of it (e.g. delay a certain flight for 2 hours).
ActionType	Enumerate type that include all of the possible actions in this model (delay, cancel, use idle aircraft and switch).
Conflict	Comprise the type of a certain conflict and the details of it (affected flights, airport and etc.). data for a certain conflict (conflict type and detail information for this conflict).
ConflictType	Enumerate type that include all of the possible conflicts in this model (overlap, noncontinuous trajectory, etc).
PriorityQueue	Support class, allows $O(1)$ access to the lowest-priority item in the queue.

Table 1: Brief description of other classes

4 RESULT

4.1 Solution to the given problem

The summary of our solution is shown in table 2, since we apply a strict rule on the 'idle' and 'switch' action, they only take a tiny share in the final solution. And given the fact that there are only 14 flights in the $8:00-10:00$ restriction period, the number of average actions it takes to address a initial conflict are 1.43. Our model tends to use delay in most scenarios, since normally two 'cancel' actions are equivalent to three 'delay' action.

Features	Total
Number of Cancelled Flight	4
Number of Delayed Flight	28
Number of Flight which the last arrival airport of it is changed	0
Number of Flight which the aircraft type of it is changed	2
Delay Time in Total (Hour)	45.067
Number of Passengers of the Cancelled Flights	417
Number of Passengers of the Delayed Flights Number of Transferred Passengers	4426
Number of Transferred Passengers	56
Reduced Turnaround Time in total (Minute)	198

Table 2: Solution impact in various aspects

4.2 Time Consumption

The average run time of our model under different initial conditions are presented in table 3, it is clear that the overall run-time is increasing while the problem went harder, since our method is basically a search algorithm. Besides, the run-time is also affected by the complexity of the inter-relationship among all flights, because the searching route can be seen as a tree, if there are too many inter-connections among all flights, the average number of children for a single parent node grows, and it will take longer for the model to get the solutions.

The total length of the no fly period (Hour)	Number of flight affected initially	Average run time (Second)
2 (8:00-10:00 28/02/2018 at AIRPORT_57)	14	24.44
4 (8:00-12:00 28/02/2018 at AIRPORT_57)	22	58.08
8 (8:00-12:00 28/02/2018 at both AIRPORT_57 and AIRPORT_50)	28	65.07

Table 3: Run time for different length of no-fly period

5 LIMITATION, FUTURE WORK AND CONCLUSION

Although we achieve a promising result with an acceptable speed, there are still some weaknesses can be overcame in the future and some limitations needs to be noticed. First, since our algorithm is a rule-based system, our solution is heavily relied on existing constraints and situations. Generalise the proposed algorithm to more fuzzy system and other scenarios can be an important future direction. Second, our search algorithm complexity is exponentially to the conflicted flights in the list. That is, the time consumption will be very large if many flights are delayed during a short period.

Nevertheless, our proposed solution still outperform many other models we tried. It takes much less time than other high computational optimisation methods such as genetic algorithm. It also achieve a good performance compared with some native methods such as Greedy algorithm.

6 ACKNOWLEDGMENT

We thank Lixia Deng from School of Science, RMIT University who provided insight that greatly assisted the research.

REFERENCES

[1] Donald B Johnson. 1973. A note on Dijkstra's shortest path algorithm. *Journal of the ACM (JACM)* 20, 3 (1973), 385–388.

[2] Zhe Liang, Fan Xiao, Xiongwen Qian, Lei Zhou, Xianfei Jin, Xuehua Lu, and Sureshan Karichery. 2018. A column generation-based heuristic for aircraft recovery problem with airport capacity constraints and maintenance flexibility. *Transportation Research Part B: Methodological* 113 (2018), 70–90.

[3] Wei Shao, Flora D Salim, Tao Gu, Ngoc-Thanh Dinh, and Jeffrey Chan. 2018. Traveling Officer Problem: Managing Car Parking Violations Efficiently Using Sensor Data. *IEEE Internet of Things Journal* 5, 2 (2018), 802–810.