THE AIRPORT GATE ASSIGNMENT PROBLEM

H. Ding[1], A. Lim[2], B. Rodrigues[3] and Y. Zhu[2]

[1]Dept of Computer Science, National University of Singapore,
3 Science Drive 2, Singapore 117543
dinghaon@comp.nus.edu.sg

[2]Dept of IEEM, Hong Kong University of Science and Technology,
Clear Water Bay, Hong Kong
{iealim,yzhu}@ust.hk

[3]School of Business, Singapore Management University,
469 Bukit Timah Road, Singapore 259756
br@smu.edu.sg

ABSTRACT

In this paper we study the over-constrained Airport Gate Assignment Problem where the objectives are to minimize the number of ungated flights and the total walking distances or connection times. We approach the problem by first using a greedy algorithm to minimize ungated flights and by developing exchange moves which then facilitate the use of heuristics. Simulated Annealing and a hybrid of Simulated Annealing and Tabu Search are used. Experimental results are good and exceed those previously obtained.

Keywords: Heuristics, Logistics, Operations Management, Optimization

INTRODUCTION

The main purpose of flight-to-gate assignments is to assign aircraft to suitable gates so that passengers can conveniently embark and disembark. The distance a passenger has to walk in any airport to reach various key areas, including departure gates, baggage belts and connecting flights provide for an important performance measure of the quality of any airport. Certain walking distances are fixed, others are dynamic. This allows for the ground handling agents and airlines, together with airport authorities, to dynamically assign airport gates to scheduled flights so as to minimize walking distances while, consequently, minimizing connection times. Which flight-to-gate assignment policy to be used so as to achieve such minimum times can be derived at the start of each planning day based on published flight schedules and booked passenger loads. The Airport Gate Assignment Problem (AGAP) seeks feasible flight-to-gate assignments so that total passenger connection times, as can be proxied by walking distances, is minimized. Planning horizons would typically be time intervals that include so-called airport peak periods since if proper gate assignments are done during such periods, gate shortages will hardly occur outside these periods. Distances that are taken into account are those from check-in to gates in the case of embarking or originating passengers, from gates to baggage claim areas (check-out) in the case of disembarking or destination passengers and from gate to gate in the case of transfer or connecting passengers. In the over-constrained case, where the number of aircraft exceeds the number of available gates, we include the distance from the apron or tarmac area to the terminal for aircraft assigned to these areas.

In the gate assignment problem, the cost associated with the placing of an aircraft at a gate depends on the distances from key facilities as well as the relations between these facilities. The basic gate assignment problem is a quadratic assignment problem and shown to

be NP-hard in Obata [5]. Babic et al.[1] formulated the gate assignment problem as a linear 0-1 IP. A branch-and-bound algorithm is used to find the optimal solution where transfer passengers are not considered. Mangoubi and Mathaisel [4] used an LP relaxation of an IP formulation and greedy heuristics solve the problem of Babic et al. with the difference that their model includes transfer passengers.

Since the gate assignment problem is NP-Hard, various heuristic approaches have been suggested by researchers. Haghani and Chen [3], proposed a heuristic that assigns successive flights parking at the same gate when there is no overlapping. In the case where there is overlapping, flights are assigned based on shortest walking distances coefficients. More recently, Xu and Bailey [6] provide a Tabu Search meta-heuristic to solve the problem. The paper [2] considers the over-constrained gate assignment problem whose objective is to minimize the number of ungated aircraft while minimizing total walking distances. In this paper, the same problem is studied but different approaches are applied.

PROBLEM DESCRIPTION AND FORMULATION

We give here a definition of the over-constrained AGAP, where the following notations are used:

N: set of flights arriving at (and/or departing from) the airport;
M: set of gates available at the airport;
n: total number of flights, i.e., $|N|$, where $|N|$ denotes the cardinality of N;
m: total number of gates, i.e., $|M|$;
a_i: arrival time of flight i;
d_i: departure time of flight i;
$w_{k,l}$: walking distance for passengers from gate k to gate l;
$f_{i,j}$: number of passengers transferring from flight i to flight j;

Additionally, we will make use of two dummy gates. Gate 0 represents the entrance or exit of the airport, and gate $m+1$ represents the apron or tarmac where flights arrive at when no gates are available.

Letting the binary variable $y_{i,k} = 1$ denote that flight i is assigned to gate k ($0 < k \leq m+1$), $y_{i,k} = 0$ otherwise. Then, the following constraint must be satisfied:

$$\forall (i,j)\ y_{i,k} = y_{j,k} = 1 (k \neq m+1)\ \textit{implies}\ a_i > d_j\ \textit{or}\ a_j > d_i. \qquad (*)$$

This condition disallows any two flights to be scheduled to the same gate simultaneously (except if they are scheduled to the apron or tarmac).

Our objectives are to minimize the number of flights assigned to the apron, and the total walking distance, which consists of three components: the walking distance of transfer passengers, disembarking arrival passengers and originating departure passengers. The formulation of the AGAP can be expressed as follows.

Minimize $\sum_{i=1}^{n} y_{i,m+1}$

Minimize $\sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{m+1} \sum_{l=1}^{m+1} f_{i,j} w_{k,l} y_{i,k} y_{j,l} + \sum_{i=1}^{n} f_{0,i} w_{0,i} + \sum_{i=1}^{n} f_{i,0} w_{i,0}$
subject to:

$$\sum_{k=1}^{m+1} y_{i,k} = 1 (\forall i, 1 \leq i \leq n) \qquad (1)$$

$$y_{i,k}y_{j,k}(d_j - a_i)(d_i - a_j) \leq 0(\forall i, j, 1 \leq i, j \leq n, k \neq m+1) \quad (2)$$

$$y_{i,k} \in \{0,1\}(\forall i, 1 \leq i \leq n, \forall k, 1 \leq k \leq m+1) \quad (3)$$

The constraint (1) ensures that every flight must be assigned to one and only one gate or assigned to the apron. The constraint (2) expresses the fact that two flight schedules cannot overlap if they are assigned to the same gate. This constraint is an alternative form to (*) given above. The basic gate assignment problem is a quadratic assignment problem and shown to be NP-hard in [5].

A SIMULATED ANNEALING APPROACH

We approach the problem by first using a greedy algorithm to minimize the number of ungated flights, then minimize the total walking distances. As the greedy algorithm has been mentioned in [2], we discuss the heuristics to minimize the total walking distances directly.

Neighborhood Search

We use a neighborhood search approach that consists of three moves, which are:

- *The Insert Move:* Move a single flight to a gate other than the one it currently assigns. This move is the same as the original insert move.
- *The Interval Exchange Move:* Exchange two flight intervals in the current assignment. A flight interval consists of one or more consecutive flights in one gate.
- *The Apron Exchange Move:* Exchange one flight which has been assigned to the apron with a flight that is assigned to a gate currently.

The *Insert Move* is trivial and has been discussed in [6] and *Apron Exchange* is also straightforward. We now discuss the *Interval Exchange Move* in greater detail.

Two exchange moves were proposed in [6], which are the single flights exchange move and the consecutive flight pairs exchange move. However, as we observed, these neighborhood moves are not very flexible and sometimes impossible to use to get to feasible solutions. As an example, consider the case shown in Figure 1. We can see that no single flights exchange or consecutive flight pairs exchange can provide feasible solutions. However, we note that the three flights in gate A can be exchanged with the two flights in gate B. This leads us to define the following move which is a generalized form of the two original exchange moves. We exchange the flights whose arrival and departure time are between flight a and b (inclusive) in gate k with the flights whose arrival and departure time are between flight c and d in gate l. This is expressed by *(a,b,k)* $\leftrightarrow$ *(c,d,l)* and illustrated in Figure 2. As flights between a,b and c,d are represented by two intervals on the axis, this method is called *Interval Exchange Move*.

We state here some of the advantages of the *Interval Exchange Move*

- It is the generalized form of *Exchange I Move* and *Exchange II Move* and can replace these two moves.
- If we say *Exchange I Move* is "1-1 Move" and *Exchange II Move* is "2-2 Move", then *Interval Exchange Move* is a "Many-Many Move", which allows moves to be more variable, and good quality solutions easier to find;
- The *Interval Exchange Move* can be applied to more diverse neighborhoods such as found in realistic situations.

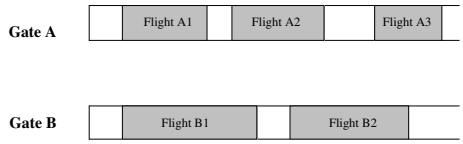

Figure 1: Example of Simple Exchange Failure

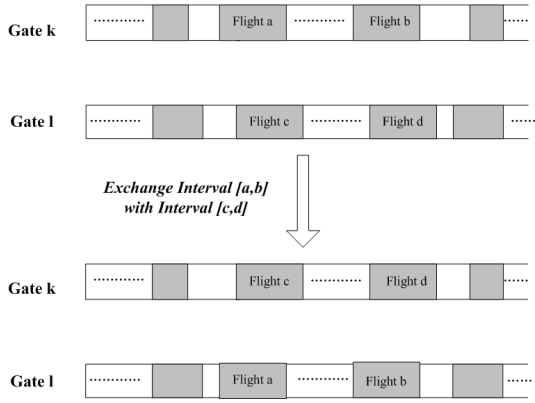

Figure 2: Interval Exchange Move

The Simulated Annealing Framework

We can now describe our SA approach. In each iteration of SA, one move is generated. The move can be of one of the three types of moves but with a certain probability in the context of SA. Each neighborhood move is accepted if it improves the objective function value. Other possible solutions are also accepted according to a probabilistic criterion. Such probabilities are based on the annealing process and they are obtained as a function of system temperature. The SA heuristic framework that we employ for the AGAP is as follows:

1. Get an initial solution x_{now} by the greedy method described above. Set $x_{best} = x_{now}$,
2. Set the annealing temperature T as a linear function to the input size, $T = T_{const} * n$, where T_{const} determines the starting temperature, and n is the number of aircraft.
3. Determine the type of neighborhood move by uniform probability. Randomly generate a neighborhood of this type and calculate the delta cost Δ if the generated neighborhood move is performed.
4. Decide whether to perform the neighborhood move generated, with the probability, $p_o = a * exp(-\Delta/(k * T))$, where constants a and k determine the accept rate. If the move is performed and the cost is smaller than x_{best}, update x_{best}
5. Decrease T by a cool rate factor d; $T = T * d$. If T does not meet the termination criteria, go to step 3

A HYBRID SIMULATED ANNEALING WITH TABU SEARCH APPROACH

We implement a hybrid algorithm using SA with the Interval exchange TS (ITS) described in [2]. In this approach, we use the same structure as that used in SA. The difference is, when the number of iterations for which the result is not improved or the neighborhood move is not accepted exceeds a certain value, we use ITS for some iterations. After ITS, we reheat and continue.

The framework is as follows:

1. Get an initial solution x_{now} by the greedy method. Set $x_{best} = x_{now}$.
2. Set the annealing temperature T as a linear function to the input size: $T = T_{const} * n$

, where T_{const} determines the starting temperature, and n is the number of aircraft. Variables *unimproved* and *unaccept* record the number of iterations for which the cost has not improved and number of iterations that no neighborhood move is performed, respectively.
3. Determine the type of neighborhood move. Randomly generate a neighborhood of the type and calculate the delta cost Δ, if the generated neighborhood move is performed.
4. Decide whether to perform the neighborhood move generated, with the probability $p_o = a * exp(-\Delta/(k * T))$, where constants a and k determine the accept rate. If the move is performed and the cost is smaller than x_{best}, update x_{best}. Update the variable *unimproved* and *unaccept*.
5. **if** ($unimproved > max\_improve$) or ($unaccept > max\_accept$) Perform a ITS as described in the previous section for a number of iterations. Reheat the temperature by a factor, $reheat$,: $T = T * reheat$; **else** Decrease the temperature by a cool rate factor, d : $T = T * d$
6. If the termination requirement is not met, return to step 3

EXPERIMENTAL RESULTS AND ANALYSIS

We implemented the SA algorithm in Java and ran them on a UNIX server SunFire. The five sets of test data used are the same as in [2]. We compare the results obtained with SA with the results obtained by the Interval Exchange TS (ITS) method given in [2] and by our Hybrid SA and TS method and by a brute force method which enumerates all the solutions. In each case, the greedy method is applied to determine the data class; for example, in Test Set 1, we use a small set of flights and gates and determine, that for this set, there is no flight that is left ungated by our greedy method. 5 groups of test instances are generated and results are discussed in the following. The detailed results are not presented here due to the space limitation.

• **Test Set 1: Small Input with no ungated flights** The first test set consists of 10 small-size inputs. All the flights can be assigned to the gates. We find the Hybrid method finds all optimal solutions that ITS and SA find. Furthermore, it takes less time than SA. Also, SA finds all optimal solutions, while ITS achieves 9 out of 10 solutions.

• **Test Set 2: Small Input with ungated flights** The second test set consists of 10 small test cases, but some flights must be assigned to the apron in each case. The ungate flights are transformed to large penalties to reflect in the objective functions.The Hybrid method finds all the optimal solutions while the running time is slightly better than the SA method. The SA method obtains only 4 of 10 of the optimal solutions, while ITS obtains 9 out of 10 of the optimal solutions. We can conclude from Test Sets 1 and 2, that the SA method does not perform well for small input size. The ITS method gets better results in a shorter period of time.

• **Test Set 3: Randomized Large Input** 100 test cases are generated in this set, and they are categorized into 10 different sizes with 10 cases in each size. We find that for all the 10 groups of data, the Hybrid method gives better results than both ITS and SA. The results of SA are not as good as that from ITS. However, we notice that the CPU time for SA is extremely low. When we attempt to increase the number of iterations, the results are not improved much.

• **Test Set 4: Fully packed Input** In this set of test cases, all the flights are "fully packed", i.e., one flight is followed by another. There are no gaps between the consecutive flights. This is an extreme case. Similar to Set 3, 10 groups of test cases are generated,

and each contains 10 cases. Hybrid method provide a better solution than SA method and SA method performs better than ITS. Clearly, the Hybrid method takes a relatively long running time for this case. Even we increase the number of iterations of SA method, we are not likely to get results as good as from the Hybrid method. We find that for fully packed test cases, SA gives better results than ITS. And for large input cases, SA's run time is much faster than TS, although it was run on a slower CPU.

• **Test Set 5: Large Input with varying densities** As in Sets 3 and 4, 100 cases are generated with 10 different sizes. The test case sizes are from 100*16 to 460*34 and the densities of flights are ranged from 55% to 95% (the density of full packed input is 100%). We find that Hybrid method gives relatively large improvement over most of the cases (9 out of 10) to ITS. Additionally, it is better in all of 10 cases than SA.

From the results, we find that SA does not have an advantage over ITS. However, it shows superiority in the running time. The SA method has the advantage that it can decrease the objective function value in a very short time, but very little more after this. Because of this, we developed the combination of SA with TS to form a Hybrid method in which SA can decrease the cost in a short time while TS can continue to improve results.

CONCLUSION

In this paper, we have applied two heuristic methods: Simulated Annealing and Hybrid Simulated Annealing with Tabu Search to the AGAP problem. The results obtained by a purely SA approach are not as good as the Interval Exchange TS (ITS) approach proposed in [2]. However, the approach has advantages in running time since it can reach a relatively low values in short times. We take advantage of this latter property and consequently developed a Hybrid method which inserts TS steps into SA, which we have found gives better results than both ITS and SA approaches and in a reasonable time.

References

[1] Babic, O., Teodorovic, D. & Tosic, V. Aircraft stand assignment to minimize walking. *Journal of Transportation Engineering, 1984, 110*, 55–66.

[2] Ding, H., Lim, A., Rodrigues, B. & Zhu, Y. New heuristics for the over-constrained airport gate assignment problem. *to appear, 2002*.

[3] Haghani, A. & ching Chen, M. Optimizing gate assignments at airport terminals. *Transportation Research A, 1998, 32*(6), 437–454.

[4] Mangoubi, R.S. ; Mathaisel, D. Optimizing gate assignments at airport terminals. *Transportation Science, 1985, 19*, 173–188.

[5] Obata, T. The quadratic assignment problem: Evaluation of exact and heuristic algorithms. *Tech. Report TRS-7901, Rensselaer Polytechnic Institute, Troy, New York, 1979*.

[6] Xu, J. & Bailey, G. The airport gate assignment problem: Mathematical model and a tabu search algorithm. In *Proceedings of the 34th Hawaii International Conference on System Sciences-2001* (2001).