Aircraft and Gate Scheduling Optimization at Airports

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

[1] *Department of CS, National University of Singapore*
3 Science Drive 2, Singapore
dinghaon@comp.nus.edu.sg

[2] *Department of IEEM, The Hong Kong University of Science and Technology*
Clear Water Bay, Hong Kong
{iealim,zhuyi}@ust.hk

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

Abstract

In this paper, we consider the over-constrained Airport Gate Assignment Problem where the number of flights exceeds the number of gates available, and where the objectives are to minimize the number of ungated flights and the total walking distances or connection times. We design a greedy algorithm and use a Tabu Search meta-heuristic to solve the problem. The greedy algorithm minimizes ungated flights while providing initial feasible solutions while we devise a new neighborhood search technique, the Interval Exchange Move, which allows us flexibility in seeking good solutions, especially in case when flight schedules are dense in time. Experiments conducted give good results.

1. Introduction

Modern airport design incorporates terminal features that facilitate good passenger handling. In particular, terminal topology from check-in counters to embarkation or disembarkation gates attempt to provide for smooth and optimal flow of passengers (Figure 1). 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 for the quality of any airport. While certain walking distances are fixed, others are dynamic. In particular, the distances traversed by passengers from check-in counters to gates and from gate to gate, in the case of transfer or connecting passengers, change according to how scheduled flights are assigned to gates. 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 to find 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 exceed the number of available gates, we include the distance from the apron or tarmac area to the terminal for aircraft assigned to these areas.

This paper is organized as follows. In the next section, we provide a short summary of previous work done

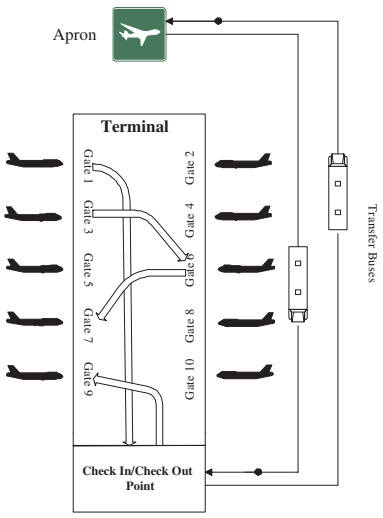

Figure 1. Airport layout and passenger flows

on this and related problems. In Section 3, we give a basic formulation of the AGAP as we model it. In Section 4, we discuss a basic greedy algorithm that minimizes the number of flights not assigned to gates and in Section 5 we give a Tabu Search Heuristic for the AGAP. In Section 6, computational results and comparisons are given and in the final Section 7, we summarize our approach and findings.

2. Previous work

One of the first attempts to use quantitative means to minimize intra-terminal travel into a design process was given by Braaksma and Shortreed [2]. The assignment of aircraft to gates, which minimize travel distances, is an easily motivated and understood problem but a difficult one to solve. The total passenger walking distance is based on passenger embarkation and disembarkation volumes, transfer passenger volumes, gate-to-gate distances, check-in-to-gate distances and aircraft-to-gate assignments. 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 [7]. 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. Network models [9] and Simulation models [3, 4] were also proposed to formulate the problem.

Since the gate assignment problem is NP-Hard, various heuristic approaches have been suggested by researchers,e.g. Haghani and Chen [6]. 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 [8] provide a Tabu Search meta-heuristic to solve the problem. The algorithm exploits the special properties of different types of neighborhood moves, and creates highly effective candidate list strategies. Although much work has been centered on the gate assignment problem with the objective of minimizing distance costs (or variants of this), to the best of our knowledge, no previous work has considered the over-constrained gate assignment problem. In particular, no previous work has addressed both the objectives of minimizing the number of ungated aircraft while minimizing total walking distances.

3. Problem definition

In this paper, we discuss the over-constrained AGAP which attempts to schedule a set of flights during any given planning day and assign them to different gates so as to minimize the number of flights that are not assigned to any gate; additionally, we seek to minimize walking distances between gates and hence minimize connection times. In our model, we will not stipulate secondary constraints which account for boarding times and other buffers between arrival and departure times since these are easily dealt with by extending flight arrival and departure durations. 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. Hence, $w_{k,0}$ represents the walking distance between gate k and the airport entrance or exit, and $f_{0,i}$ represents the number of

originating departure passengers of flight i; $f_{i,0}$ represents number of the disembarking arrival passengers of flight i. So $w_{m+1,k}$ represents the walking distance between the apron and gate k (usually much larger than the distance among different gates).

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:

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

This condition disallows any two flight 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:

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

$$a_i < d_i(\forall i, 1 \leq i \leq n) \quad (2)$$

$$y_{i,k}y_{j,k}(d_j - a_i)(d_i - a_j) \leq 0(\forall i,j, 1 \leq i,j \leq n, k \neq m+1) \quad (3)$$

$$y_{i,k} \in \{0,1\}(\forall i, 1 \leq i \leq n, \forall k, 1 \leq k \leq m+1) \quad (4)$$

The constraint (1) ensures that every flight must be assigned to one and only one gate or assigned to the apron. The constraint (2) specifies that each flight's departure time is later than its arrival time. The constraint (3) says that two flights' schedule cannot overlap if they are assigned to the same gate. Actually, the last constraint can be expressed in an alternative form as condition (*) given above.

4. A greedy algorithm for minimizing the number of flights assigned to the apron

To solve the over-constrained AGAP, a first step is to minimize the number of flights that need be assigned to the apron. The minimal number of flights can be calculated by the following greedy algorithm.

The basic idea of the greedy algorithm is as follows. After sorting all the flights by the departure time, flights are assigned to the gates one by one. Any flight will be assigned to an available gate with latest departure time. If there are no gates available, the flight will be assigned to the apron. This is illustrated in Figure 2.

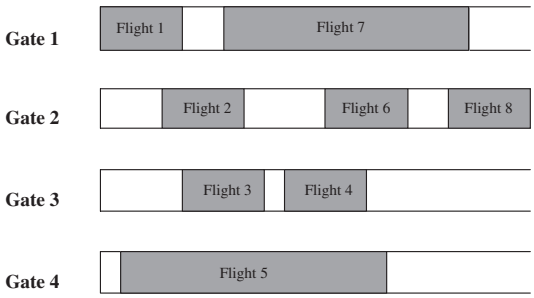

Figure 2. A greedy algorithm for minimizing the number of flights assigned to the apron

The basic details of the algorithm are as follows.

1. Sort the flights according to the departure time $d_i(1 \leq i \leq n)$. Let $g_k(1 \leq k \leq m)$ represents the earliest available time (actually the departure time of last flight) of gate k. Set g_k=-1 for all k.
2. For each flight i
   (a) Find gate k such that $g_k < a_i$ and g_k is maximized;
   (b) If such k exists, assign flight i to gate k, update g_k=d_i;
   (c) If k does not exist, assign flight i to the apron.
3. Output the result.

Proof of the correctness of the greedy algorithm: By induction, assume we have found the optimal solution after scheduling flight i by the greedy algorithm. Now by this, we will assign flight f to gate k. But the optimal solution is to drop flight f and assign $f'(f' > f)$ to gate k'. Hence we can always replace f' by f to make our greedy solution no worse than the optimal solution.

There are two cases we should consider.

1. If $k = k'$, since we sort the flights by departure time, $d_f \leq d_{f'}$. We have $g_k \leq g'_k$. As we considered the earliest available time of the gates, we find the greedy solution is better or at least equal to the optimal solution.
2. If $k \neq k'$, we find that $g_k \leq g'_{k'}$, and $g_{k'} \leq g'_k$, since we choose the maximum g_k in the greedy solution. Figure 3 illustrates this.

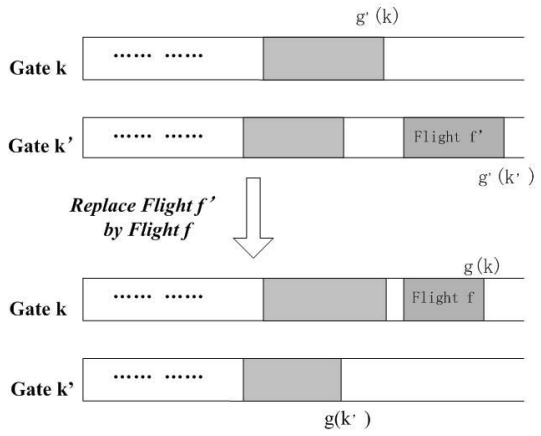

Figure 3. The correctness of the greedy algorithm

Hence the greedy method yields the optimal solution.

The greedy solution not only gives us the optimal number of flights that can be scheduled in gates, but also helps us to get a feasible initial solution, which will be used in the heuristic algorithm we discuss next.

5. Tabu search heuristic

5.1. analysis of the current approach

In the only paper on the AGAP or its variants we have found which has used meta-heuristics, Xu and Bailey [8] have proposed a Tabu Search (TS) heuristic to solve the AGAP. In their work, three neighborhood moves, namely *Insert Move*, *Exchange I Move* and *Exchange II Move* were adopted.

There are, however, some shortcomings in the approach given by Xu and Bailey, which include the following:

1. The method cannot handle over-constrained situations when there is a need assign some flights to the apron;
2. The method chooses an initial solution by a random assignment, therefore no feasible initial solution is guaranteed, though feasible solutions exist;
3. The two exchange moves are inflexible, especially Exchange II Moves; consequently, it is not easy to find good quality moves when flights schedules are dense in time.

As we can easily verify, the first two shortcomings (1), (2), can be directly resolved by the greedy algorithm provided here. The last, (3), can be tackled by an *Interval Exchange Move*, which we will present in the following section.

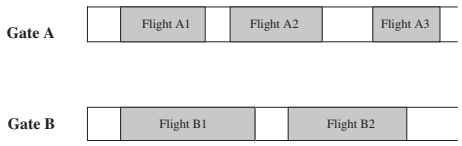

Figure 4. The situation where the simple exchange methods fails

5.2. New neighborhood search methods

We will use a new 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 [8]. We now discuss the *Interval Exchange Move* and *Apron Exchange Move* in greater detail.

5.2.1. The interval exchange move As mentioned, two exchange moves were proposed in [8], which are single flights exchange move and 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 4.

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 $(a,b,k) \leftrightarrow$

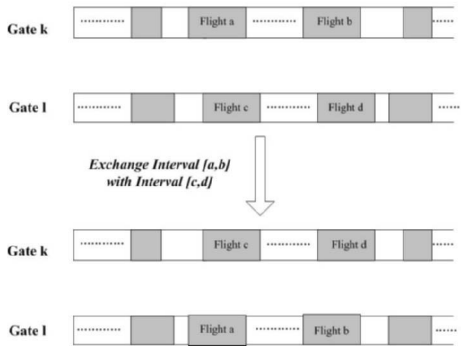

Figure 5. Interval exchange move

Figure 6. The four time points of an interval

(c,d,l) and illustrated in Figure 5. As flights between a,b and c,d are represented by two intervals on the axis, this method is called *Interval Exchange Move*.

Now, the essential reason for the *Interval Exchange Move* is to find two compatible intervals, which will allow us to get a feasible solution. In order to get this, "interval" data should contain four time points: the earliest available time ($t1$), the start time ($t2$), the end time ($t3$) and the latest available time ($t4$). Figure 6 illustrates the meaning of these four time points.

Further to this, we define two functions on intervals. *ExtendLeft()* extends the current interval by adding the flight which is just left to it, and *ExtendRight()* extends the current interval by adding the flight which is just right to it. The functions return Boolean values to indicate whether the operations are successful. For example, the *ExtendLeft()* operation will fail if the current interval has included the first flight. Additionally, *Previous(i)* returns the flight just arranged before flight i in the same gate, *Next(i)* returns the flight just arranged after flight i.

With these, we can now state an algorithm for finding compatible intervals in Algorithm 1.

It is clear that after finding two compatible intervals, exchange and update can be done easily.

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

Algorithm 1 finding compatible intervals

1: Select two flights a,b in different gates, where a,b have overlap time
2: Initialize interval $A \leftarrow a$, interval $B \leftarrow b$;
3: $A.t1 \leftarrow Previous(a).departure$;
4: $A.t2 \leftarrow a.arrival$;
5: $A.t3 \leftarrow a.departure$;
6: $A.t4 \leftarrow Next(a).arrival$;
7: $B.t1 \leftarrow Previous(b).departure$;
8: $B.t2 \leftarrow b.arrival$;
9: $B.t3 \leftarrow b.departure$;
10: $B.t4 \leftarrow Next(b).arrival$;
11: $success \leftarrow true$;
12: **while** A and B are incompatible and $success$ is true **do**
13: if ($A.t2 < B.t1$ and $!ExtendLeft(B)$)
14: $success \leftarrow false$;
15: if ($B.t2 < A.t1$ and $!ExtendLeft(A)$)
16: $success \leftarrow false$;
17: if ($A.t3 > B.t4$ and $!ExtendRight(B)$)
18: $success \leftarrow false$;
19: if ($B.t3 > A.t4$ and $!ExtendRight(A)$)
20: $success \leftarrow false$;
21: **end while**
22: if ($success$)
23: exchange interval A and B;
24: else output "Exchange Failed";

- 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 (see also in Figure 4).

5.2.2. The apron exchange move The *Apron Exchange Move* is used to deal with the flights that are assigned to the apron. In each move, we exchange one flight that is assigned to the apron currently with a flight that has been assigned to a gate. As the minimal number of flights out of the gates has been determined by the greedy algorithm, we cannot perform a "many-many" exchange, so we can only effect a single flight exchange.

5.3. Tabu short-term memory

Tabu Search memory plays an important role in the search process. It forbids the solution attribute changes

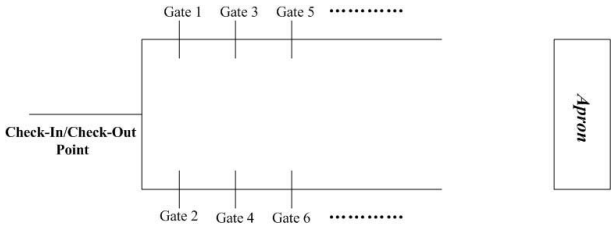

Figure 7. The layout of the airport in test data

recorded in the short-term memory to be reused. How long a restriction is in effect depends on the *tabu tenure* parameter, which identifies the number of iterations a particular restriction remains in force [5].

In our AGAP problem, since there are three types of neighborhood search moves, the tabu short-term memory can be implemented as follows (where *iter* denotes the current iteration number):

1. *Insert Move*: denoted as $(i,k) \rightarrow (i,l)$, $tabu((i,k) \rightarrow (i,l)) = iter + tabu\_tenure$ — to prevent the move $(i,l) \rightarrow (i,k)$;
2. *Interval Exchange Move*: denoted as $(a,b,k) \leftrightarrow (c,d,l)$, $tabu((a,b,k) \leftrightarrow (c,d,l)) = iter + tabu\_tenure$ — to prevent the move $(a,b,l) \leftrightarrow (c,d,k)$;
3. *Apron Exchange Move*: denoted as $(a,k) \leftrightarrow (b,OUT)$, $tabu((a,k) \leftrightarrow (b,OUT)) = iter + tabu\_tenure$ — to prevent the move $(a,OUT) \leftrightarrow (b,k)$.

6. Experimental results

We conducted elaborate experiments to compare our TS heuristic with other such approaches, specifically, with results from the approach of [8]. In the latter, there is one objective to minimise cost, whereas we are interested, additionally, in minimising the number of displaced aircraft. Moreover, their model incorporates additional constraints such as boarding time and other time buffers. We realized, however, that these latter constraints can be easily dealt with by extending flight durations. More importantly, however, Xu and Bailey focus on TS heuristics for application to the AGAP and it is this approach that we make comparisons with here.

In this section, we first explain how the test data is generated and then provide details of results and analysis.

6.1. Test data generation

We take a representative layout of an airport to have two parallel sets of terminals, where gates are symmetrically located in the two terminals shown in Figure 7. We set the distance between the check-in / check-out point to gate 1(and gate 2) to be 5 units; the distance between two adjacent gates in one terminal (e.g., gate 1 and gate 3) to be 1 unit; and the distance between two parallel gates in different terminals (e.g., gate 1 and gate 2) to be 3 units. To simplify the problem, we assume that the passengers can only walk "horizontally" or "vertically", i.e., if one passenger wants to transfer from gate 3 to gate 2, his walking distance is 1+3=4 units (The distance measure is known as *Manhatten Distance*).

The arrival time a_i of flight i is randomly generated in the interval $[10i, 10i+7]$, and the departure time d_i is generated in the interval $[a_i + 60, a_i + 69]$.

we consider the following realistic scenarios to generate $f_{i,j}$, the number of transferring passengers between flight i and flight j,

- The total number of passengers in a flight is usually within a certain interval, say [300,400];
- There are rarely very small numbers of passengers transferring from one flight to another flight; it is usually in larger numbers as in "group" transfers;
- The number of transfer passengers will increase if flight schedules are close, but not too close.

Furthermore, for each flight i, the number of disembarking passengers whose destination is this airport $f_{i,0}$ and the number of embarking passengers whose origin is this airport $f_{0,i}$ are both generated within the interval [1,100].

6.2. Parameters settings

There are a few parameters which should be set for our TS algorithm which we call the Interval Exchange TS (ITS) and the TS algorithm in [8] which we call Xu and Bailey's TS (XTS). In view for comparison, most parameters and moves in the two algorithms are the same as described in [8], except for the following:

- The iteration number in XTS is $40m-500$, the minimal iteration number is 500;
- The iteration number in ITS is $300m - 400$, the minimal iteration number is 2000, as ITS runs faster than XTS;
- The *Exchange I Move* and *Exchange II Move* in XTS are replaced by *Interval Exchange* and *Apron Exchange* in ITS respectively.

6.3. Results and analysis

We implement XTS and ITS algorithms in JAVA and run them in PIII667 Windows machine. In order to compare their effectiveness in the situation that some flights should be assigned to the apron, we have added the *Apron Exchange Move* to XTS. This will not affect the performance two exchange moves of XTS since these only deal with the flights that are assigned to gates. We also code a brute-force method to get the optimal solution for small input size test cases. Five different types of test data are generated and the details of the results and analysis are presented in the following sections.

6.3.1. Test set 1: small inputs without flights out of gates The first test set consists of 10 small size inputs. All the flights can be assigned to the gates. We runs the test case using brute-force method, XTS and ITS, and the results are shown in Table 1.

We observe that, in small input test cases, ITS can get the optimal solutions in most of the time (9 out of 10); however, XTS can only achieve 3 optimal solutions. Both heuristics runs much faster than the exact method in the relatively larger cases.

6.3.2. Test Set 2: small inputs with flights out of gates The second test set also consists of 10 small test cases, but some flights should be assigned to the apron in these cases. The results are presented in Table 2.

We find that ITS is still superior. It achieves optimal solutions among all the cases. XTS obtains 4 out of 10. Also, the running time of ITS is good.

6.3.3. Test Set 3: randomized large inputs 100 test cases are generated in this set, and they have 10 different sizes, 10 cases in each size. We compare ITS and XTS with the respect to the result and running time. The details are presented in Table 3.(In all the tables, CPU time is measured in seconds.)

We find that among these 10 groups of test cases, ITS performs better than XTS in all the groups, in almost the same (or less) running time.

6.3.4. Test Set 4: fully-packed inputs In this set of test cases, all the flights are "fully packed", one flight is just followed by another. There are no gaps between the consecutive flights.

This special designed set is used to test how our interval exchange method works. Similar to Set 3, 10 groups of test cases are generated, and each contains 10 cases. Table 4 shows the results.

We find that for fully packed test cases, the advantage of ITS is much more obvious, since the single exchange moves cannot be performed in many situations. The interval exchange, however, is not similarly handicapped.

6.3.5. Test Set 5: large inputs with different densities To discover the trend of the performance with different test case sizes and flight densities, 100 cases are generated for the last set with 10 different sizes and 10 different densities. The test case sizes are from 100×16 to 460×34, and the densities are evenly distributed from 55% to 95% (100% is the fully packed case). These test results are shown in Table 5 (grouped by size) and Table 6 (grouped by density) respectively.

Although there are some exceptional cases, in general, we have found that the performance of ITS is better when density and size increase. As we know, when size and density is larger, feasible solutions that can be obtained by *Exchange I Move* and *Exchange II Move* in XTS will be less likely, and the search, consequently, easily trapped. However, the *Interval Exchange Move* will not be impeded in ITS since it is more flexible. These experimental results clearly show the advantage of the ITS method, and particularly, of the Interval Exchange Move.

7. Conclusion

In this paper, we considered the basic over-constrained AGAP, which is to minimize the number of flights assigned to the apron while minimizing the total walking distances. We provided a greedy algorithm that can allocate the flights to minimize the number of flights that will be ungated as well as to provide an initial feasible solution. We then proposed a Tabu Search algorithm with a new neighborhood search technique, the Interval Exchange Move, which is more flexible and more general than previously employed exchange moves used for this problem. This search move allows us to find good quality solutions more effectively for more diverse neighborhoods, where currently known exchange moves would not work. Experiments were conducted using a range of test data sets. Our algorithm is compared with a previous Tabu Search method, adapted for over-constrained AGAP, and results show its superiority in the AGAP optimal assignments and in running times, especially for the larger and denser data sets.

References

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

[2] J. Braaksma and J. Shortreed. Improving airport gate usage with critical path method. *Transportation Engineering Journal of ASCE 97*, pages 187–203, 1971.

Size	Brute Force		XTS		ITS	
$(n \times m)$	cost	CPU	cost	CPU	cost	CPU
15×3	7322	0.05	7591	0.451	7322*	0.63
16×3	8759	0.06	8793	0.531	8759*	0.68
17×3	9448	0.05	9538	0.561	9448*	0.711
18×4	9053	16.995	9053*	0.64	9053*	0.991
18×4	10308	0.17	10820	0.631	10308*	0.871
20×5	12153	184.195	12153*	0.761	12153*	1.192
20×5	12950	107.365	12958	0.741	12950*	1.161
20×6	13095	14469.186	13095*	0.781	13095*	1.152
22×5	11822	402.388	11856	0.081	11835	1.332
25×5	15715	1445.388	16133	1.012	15715*	1.572

* *indicates the optimal solutions*

Table 1. Results for test set 1

Size	Brute Force		XTS		ITS	
$(n \times m)$	cost	CPU	cost	CPU	cost	CPU
15×3	342690	0.19	342716	1.051	342690*	0.481
16×3	344811	0.12	344811*	0.992	344811*	0.842
17×3	325795	0.09	325795*	0.981	325795*	0.851
20×3	441774	1.071	441924	1.072	441774*	1.002
20×3	524798	0.331	524868	1.141	524798*	0.942
24×3	790815	6.81	790815*	1.122	790815*	1.102
25×4	1055866	89.82	1055866*	1.052	1055866*	1.082
25×4	1013013	150.67	1313482	1.212	1013013*	1.222
25×4	1213690	179.51	1214073	1.172	1213690*	1.132
25×4	1189388	128.69	1189630	1.102	1189388*	1.042

* *indicates the optimal solutions*

Table 2. Results for test set 2

Size	XTS		ITS		Savings
$(n \times m)$	cost	CPU	cost	CPU	(%)
100×16	106553	8.249	105451	9.092	1.035
160×20	183208	22.054	180338	22.631	1.566
220×24	252150	47.646	248361	43.962	1.503
280×28	318505	84.645	314488	75.286	1.261
340×32	390228	134.582	382374	119.796	2.013
400×36	457218	212.773	445238	178.215	2.620
460×40	528817	333.776	517772	250.532	2.089
520×44	596820	524.734	582537	347.864	2.393
580×48	664954	813.155	651299	464.284	2.053
640×52	729114	1118.897	714990	587.811	1.937

Table 3. Results for test set 3

Size	XTS		ITS		Savings
$(n \times m)$	cost	CPU	cost	CPU	(%)
100×16	228929	10.676	211500	16.675	7.613
140×18	400960	19.527	357701	22.008	10.789
180×20	556958	29.091	496218	36.866	10.906
220×22	734149	42.200	644356	60.480	12.231
260×24	934664	58.508	811147	68.162	13.215
300×26	1152765	79.762	1006603	69.511	12.679
340×28	1387079	111.708	1194938	96.902	13.852
380×30	1620387	153.561	1401163	130.030	13.529
420×32	1888442	199.746	1619008	169.448	14.268
460×34	2157675	260.184	1833132	218.553	15.041

Table 4. Results for test set 4

Size	XTS		ITS		Savings
$(n \times m)$	cost	CPU	cost	CPU	(%)
100×16	185493	7.842	181632	8.134	2.081
140×18	310932	14.138	298082	14.141	4.133
180×20	432695	21.574	412614	20.680	4.641
220×22	562638	30.873	533250	27.872	5.223
260×24	710704	41.782	670608	38.662	5.642
300×26	865970	56.050	813462	47.470	6.063
340×28	1026543	80.582	948443	83.156	7.608
380×30	1212541	110.537	1122868	111.835	7.395
420×32	1389721	144.286	1280326	144.766	7.872
460×34	1580388	184.505	1462915	174.321	7.433

Table 5. Results for test set 5 — grouped by size

Density(%)	50	55	60	65	70
Avg Savings(%)	2.873	3.226	3.698	3.565	3.513
Density(%)	75	80	85	90	95
Avg Savings(%)	5.043	4.928	5.261	8.756	13.200

Table 6. Result for test set 5 — grouped by density

[3] Y. Cheng. Network-based simulation of aircraft at gates in airport terminals. *Journal of Transportation Engineering*, pages 188–196, 1998.

[4] Y. Cheng. A rule-based reactive model for the simulation of aircraft on airport gates. *Knowledge-based Systems*, 10:225–236, 1998.

[5] F. Glover and M. Laguna. *Tabu Search*. Kluwer Acadamic Publishers, 1997.

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

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

[8] J. Xu and G. Bailey. 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.

[9] S. Yan and C.-M. Chang. A network model for gate assignment. *Journal of Advanced Transportation*, 32(2):176–189, 1998.