Available online at www.sciencedirect.com
SCIENCE DIRECT®
Computers & Operations Research 32 (2005) 1867–1880
computers & operations research
www.elsevier.com/locate/dsw

# The over-constrained airport gate assignment problem

H. Ding[a], A. Lim[b], B. Rodrigues[c,*], Y. Zhu[b]

[a] *Department of Computer Science, National University of Singapore, 3 Science Drive 2, Singapore 117543, Singapore*
[b] *Department of IEEM, Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong*
[c] *School of Business, Singapore Management University, 469 Bukit Timah Road, Singapore 259756, Singapore*

**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 total walking distances or connection times. We first use a greedy algorithm to minimize ungated flights. Exchange moves are employed to 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:* Airlines; Transportation; Tabu search; Simulated annealing

## 1. Introduction

The primary purpose of flight-to-gate assignments in airports is to assign aircraft to gates to meet operational requirements while minimizing inconveniences to passengers. Planners seek to minimize distances passengers have to walk to departure gates, baggage belts and connecting flights since this is a key quality performance measure of any airport. For connecting flights, minimizing distances is key to smooth operations due to short connection times common in many international airports. While certain walking distances are fixed when schedules can be conformed to, others can change from time to time. For example, the distances traversed by transfer or connecting passengers from gate to gate can change due to changing gate assignments resulting from randomness in operations. Airlines, ground-handling agents and airport authorities, therefore need to assign gates to flights dynamically to minimize walking distances and consequently, connection times. A flight-to-gate assignment policy satisfying operational requirements can be derived at the start of each planning day based on published flight schedules and booked passenger loads. In the Airport Gate Assignment

* Corresponding author. Tel.: +65-6822-0709; fax: +65-6822-0777.
*E-mail address:* br@smu.edu.sg (B. Rodrigues).

0305-0548/$ - see front matter © 2003 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2003.12.003

Problem (AGAP), the objective is to find feasible flight-to-gate assignments which minimizes total passenger walking distances including distances between connecting flights. In the problem, the planning horizon would typically be a time interval that includes flight peak periods since if gate assignments are done well during these periods, gate shortages are unlikely to occur outside these periods. The typical distances in airports considered are: (1) the distance from check-in to gates for embarking or originating passengers, (2) the distance from gates to baggage claim areas (check-out) for disembarking or destination passengers, and (3) the distance from gate to gate for 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.

This paper is organized as follows. In the next section, we provide a short summary of work done on this and related problems. In Section 3, we give a model for the AGAP. 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 simulated annealing heuristic for the problem. In Section 6, we provide a tabu search method and in Section 7 develop a hybrid simulated annealing with tabu search heuristic. In Section 8, computational results and comparisons are given and in Section 9, we summarize our findings and suggest future work.

## 2. Previous work

Braaksma and Shortreed [1] provided one of the first attempts to use quantitative means to minimize intra-terminal travel through the design of terminals. The assignment of aircraft to gates, which minimize travel distances, is easily motivated and understood problem but difficult 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. 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 was shown to be NP-hard by Obata [2]. Babic et al. [3] formulated the gate assignment problem as a 0–1 integer program and used a branch-and-bound algorithm to find solutions where transfer passengers were not considered. Mangoubi and Mathaisel [4] used an LP relaxation of an integer program formulation with greedy heuristics to solve the problem of Babic et al. with the difference that their model includes transfer passengers. Bihr [5] used a 0–1 LP to solve the minimum walking distance problem for fixed arrivals in a hub operation where a simplifying assignment formulation is employed. Wirasinghe and Bandara [6] considered, additionally, the cost of delays and used an approximation procedure in their analysis. Yan and Chang [7] proposed a network model that is formulated as a multi-commodity network flow problem. An algorithm based on the Lagrangian relaxation, with subgradient methods together with a shortest path algorithm and a Lagrangian heuristic was developed for the problem. Baron [8] used simulation to analyze the effects of passenger walking distance resulting from different gate-use strategies where both local and transfer passengers are considered. Simulation models for the problem can also be found in, for example, Cheng [9,10]. Some models have suggested ways to resolve stochastic flight delays in the planning stages. In Hassounah and Steuart [11], planned scheduled buffer times were found to improve punctuality. Yan and Chang [7] and Yan and Hou [12] added a fixed buffer time

between flights assigned to a gate. The gate assignment with time windows is studied in [13]. Other research on the AGAP include [14–16].

Because the gate assignment problem is NP-Hard, heuristic approaches have been used by researchers. Haghani and Chen [17], 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 [18] provided 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 except [19] has addressed both the objectives of minimizing the number of ungated aircraft and minimizing total walking distances.

## 3. Problem description and formulation

In this section, following [19], we provide a model for the over-constrained AGAP which attempts to assign flights to gates to minimize the number of flights that are not assigned to any gate; additionally, we a second objective for the problem is to minimize walking distances between gates. In the model, we do not address 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. 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 use two dummy gates, 0 and $m+1$ for the problem, where 0 represents the entrance or exit of the airport, assumed to be one point, and gate $m+1$ represents the apron or tarmac where flights arrive at when no gates are available, also assumed to be single point. Hence, $w_{k,0}$ will represent the distance between gate $k$ and the airport entrance or exit, and $f_{0,i}$ represents the number of originating departing passengers of flight $i$; $f_{i,0}$ represents number of the disembarking (arrival) passengers of flight $i$, and $w_{m+1,k}$ represents the walking distance between the apron and gate $k$ (usually much larger than the distance among different gates).

We take $y_{i,k}=1$ to denote that flight $i$ is assigned to gate $k$ $(0<k\leqslant m+1)$, $y_{i,k}=0$ otherwise. For this binary variable, the following constraint must hold, which disallows any two flights to be assigned to the same gate simultaneously (except if they are scheduled to the apron or tarmac):

$$\text{for all } (i,j) \text{ and } k\neq m+1,\ y_{i,k}=y_{j,k}=1 \Rightarrow a_i>d_j \text{ or } a_j>d_i. \qquad (*)$$

Our objectives are to minimize the number of flights assigned to the apron, i.e. those left ungated and to minimize total distances: the walking distances of embarking passengers, disembarking passengers and transfer passengers. The AGAP is as follows:

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

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

**subject to** :

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

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

$$y_{i,k} \in \{0,1\} (1 \leqslant i \leqslant n, \quad 1 \leqslant k \leqslant m+1). \tag{3}$$

Constraint (1) ensures that every flight must be assigned to one and only one gate or assigned to the apron and constraint (2) requires that flights cannot overlap if they are assigned the same gate. Constraint (3) is an alternative of (∗) given above.

## 4. A greedy method for minimizing flights ungated flights

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 optimally by a greedy algorithm which is used in [19] and which we describe here for easy reference.

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 is assigned to an available gate with latest departure time. If no gates available, the flight is assigned to the apron. The basic steps are:

1. Sort flights by departure times $d_i (1 \leqslant i \leqslant n)$. Let $g_k (1 \leqslant k \leqslant m)$ represents the earliest available time (the departure time of last flight) of gate $k$. Set $g_k = -1$ for all $k$.
2. For each flight $i$,
   find gate $k$ such that $g_k < a_i$ and $g_k$ is maximized
   if such $k$ exists, assign flight $i$ to gate $k$, update $g_k = d_i$
   if $k$ does not exist, assign flight $i$ to the apron.

## 5. A simulated annealing approach

Simulated annealing (SA), introduced by Kirkpatrick et al. [20], has been widely used in optimization where it has been effective. For the AGAP, we apply SA using the neighborhood search method developed in [19] which we summarize here.

### 5.1. Neighborhood search

Xu and Bailey [18] used three neighborhood moves, which prevail in AGAP applications, which are the following:

- *Insert Move*: Move a single flight to a gate other than the one it currently assigns.
- *Exchange I Move*: Exchange two flights and their gate assignments.
- *Exchange II Move*: Exchange two flight pairs in the current assignment. The two flights in the flight pair have to be consecutive.

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

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

As is easily verified, (1) and (2), can be resolved directly by the greedy algorithm already provided here. (3) is addressed by a new *Interval Exchange Move*, which we describe in the following section.

#### 5.1.1. 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 basic and has been discussed in [18]. We describe the *Interval Exchange Move* and *Apron Exchange Move* in greater detail.

#### 5.1.2. The Interval Exchange Move

As mentioned, two exchange moves were proposed in [18], which are the single flight exchange move and the consecutive flight pairs exchange move. However, as observed, these moves are not flexible and are sometimes impossible to use to obtain feasible solutions. As an example, consider the case shown in Fig. 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 flights whose arrival and departure time are between flight

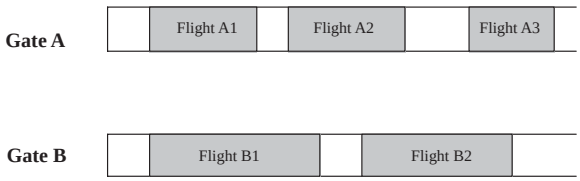


Fig. 1. Example of simple exchange failure.

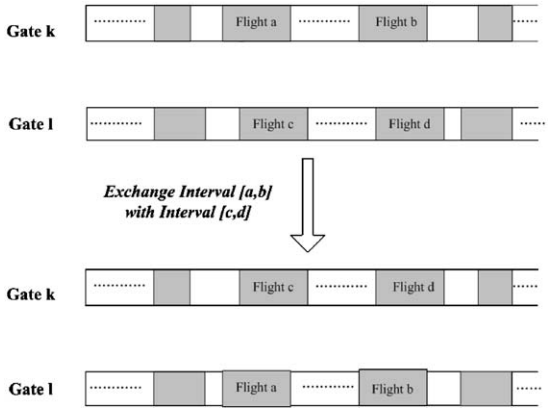


Fig. 2. Interval Exchange Move.

$a$ 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 Fig. 2. Since flights between $a$, $b$ and $c$, $d$ are represented by two intervals on the axis, this is called *Interval Exchange Move*.

The *Interval Exchange Move* has the following advantages:

- It is the generalized form of *Exchange I Move* and *Exchange II Move* and can replace these two moves.
- If the *Exchange I Move* is a "1–1 move" and the *Exchange II Move* is a "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 as found in realistic situations.

### *5.1.3. The Apron Exchange Move*

The *Apron Exchange Move* is used to deal with the flights 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 left ungated has been determined by the greedy algorithm, we do not perform a "many–many" exchange and can only effect a single flight exchange.

### 5.2. The simulated annealing framework

We can now describe the SA approach. In each iteration of SA, one move is generated. The move can be of one of three types of moves with a certain probability. 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 are obtained as a function of system temperature.

The SA heuristic framework that we employed for the AGAP is as follows:

1. Get an initial solution $x_{\text{now}}$ by the greedy method described above. Set $x_{\text{best}} = x_{\text{now}}$,
2. Set the annealing temperature $T$ as a linear function to the input size, $T = T_{\text{const}} * n$, where $T_{\text{const}}$ 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 $\Delta$ 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 $x_{\text{best}}$, update $x_{\text{best}}$
5. Decrease $T$ by a cool rate factor $d$; $T = T * d$. If $T$ does not meet the termination criteria, go to step 3.

#### 5.2.1. Parameter settings

In the above framework, we use the following constants: $T_{\text{const}}, a, k, d$

- $T_{\text{const}}$ is set to 1.95.
- $a, k$: $a$ is set to 2 and $k$ is set to 2.25.
- $d$ is set to 0.9999.
- the termination criteria is taken to be $T \leqslant 0.01$.

## 6. Interval exchange tabu search

In the AGAP problem, since there are three neighborhood search moves, tabu short-term memory can be implemented as follows, where *iter* denotes the current iteration number (see [19] where this is first implemented):

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

A TS restriction is overridden by the *aspiration criteria* if the outcome of the move under consideration is sufficiently desirable. Here, we simply define the *aspiration criteria* as, if satisfied, would mean that the candidate move would lead to a new best solution.

We can now describe a TS heuristic for the AGAP as follows:

1. Get an initial solution $x_{now}$ by the greedy algorithm. Set $x_{best} = x_{now}$ and $iter = 0$.
2. If $iter > max\_iter$, terminate with $x_{best}$. Otherwise, go to 3.
3. Determine the type of neighborhood move. Generate the neighborhood set $N(x_{now})$. Evaluate each candidate solution $x_{trial}$ as $f(x_{trial})$ for some evaluation function $f$. If $f(x_{trial}) < f(x_{best})$, accept and set $x_{next} = x_{trial}$ (aspiration criteria). Otherwise, select $x_{next} = min_{x_{trial} \in N(x_{now})} f(x_{trial})$, and $iter > tabu(x_{now} \to x_{trial})$.
4. Update the tabu memory by $tabu(x_{now} \to x_{next}) = iter + U(a, b)$. $U(a, b)$ denotes a number generated randomly between $a$ and $b$. Set $x_{now} = x_{next}$. If $f(x_{now}) < f(x_{best})$, set $f(x_{best}) = f(x_{now})$. Increase $iter = iter + 1$; go to 2.

## 7. A hybrid simulated annealing with tabu search approach

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

The hybrid 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 $\Delta$, 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 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.

### 7.1. Parameter settings

In the above framework, we use the following is used:

- $T_{const}$ is set to 2.
- $a, k$: $a$ is set to 2.25 and $k$ is set to 0.5.
- *reheat* is set to 1.25.
- $d$ is set to 0.998.
- the termination criteria: $T \leqslant 0.001$.

Table 1
Results for Test Set 1

| Size ($n \times m$) | Brute force | | ITS | | SA | | | Hybrid SA with TS | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Cost | CPU (s) | Cost | CPU (s) | Cost | CPU (s) | vs. ITS(%) | Cost | CPU (s) | vs. ITS(%) | vs. SA(%) |
| 15 × 3 | 7322 | 0.05 | 7322 | 0.63 | 7322 | 11.25 | 0 | 7322 | 17.61 | 0 | 0 |
| 16 × 3 | 8759 | 0.06 | 8759 | 0.68 | 8759 | 10.47 | 0 | 8759 | 17.94 | 0 | 0 |
| 17 × 3 | 9448 | 0.05 | 9448 | 0.71 | 9448 | 12.37 | 0 | 9448 | 17.05 | 0 | 0 |
| 18 × 4 | 9053 | 17 | 9053 | 0.99 | 9053 | 8.78 | 0 | 9053 | 8.27 | 0 | 0 |
| 18 × 4 | 10 308 | 0.17 | 10 308 | 0.87 | 10308 | 13.26 | 0 | 10 308 | 18.9 | 0 | 0 |
| 20 × 5 | 12 153 | 184.2 | 12 153 | 1.19 | 12153 | 9.23 | 0 | 12 153 | 13.19 | 0 | 0 |
| 20 × 5 | 12 950 | 107.37 | 12 950 | 1.16 | 12950 | 8.54 | 0 | 12 950 | 9.73 | 0 | 0 |
| 20 × 6 | 13 095 | 14469.2 | 13 095 | 1.15 | 13095 | 7.93 | 0 | 13 095 | 7.45 | 0 | 0 |
| 22 × 5 | 11 822 | 402.39 | 11 835 | 1.33 | 11822 | 11.1 | 0.11 | 11 822 | 13.31 | 0.11 | 0 |
| 25 × 5 | 15 715 | 1445.39 | 15 715 | 1.57 | 15715 | 13.54 | 0 | 15 715 | 14.35 | 0 | 0 |

## 8. Experimental results and analysis

We implemented the SA algorithm in Java and ran it on a UNIX server SunFire. The five sets of test data used are the same as in [19]. We compared results obtained with SA with the results obtained by the Interval Exchange TS (ITS) method given in [19] and by our hybrid SA and TS method, and by a brute force method which enumerates all 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.

### *8.1. 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 gates. The results are shown in Table 1.

From the table, we find the hybrid method finds all optimal solutions that ITS and SA find. Further, it takes less time than SA. Also, SA finds all optimal solutions, while ITS achieves 9 out of 10 solutions

### *8.2. 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 ungated flights are given large penalties which are reflected in the objective functions. The results are presented in Table 2. The hybrid method finds all the optimal solutions while the running time is slightly better than the SA method. We can see that the SA method obtains only 4 of 10 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 and the ITS method gets better results in shorter times.

Table 2
Results for Test Set 2

| Size ($n \times m$) | Brute force | | ITS | | SA | | | Hybrid SA with TS | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Cost | CPU (s) | Cost | CPU (s) | Cost | CPU (s) | vs. ITS(%) | Cost | CPU (s) | vs. ITS(%) | vs. SA(%) |
| 15 × 3 | 342 690 | 0.19 | 342 690 | 0.48 | 342 690 | 14.26 | 0 | 342 690 | 15.28 | 0 | 0 |
| 16 × 3 | 344 811 | 0.12 | 344 811 | 0.84 | 344 811 | 13.7 | 0 | 344 811 | 11.11 | 0 | 0 |
| 17 × 3 | 325 765 | 0.09 | 325 765 | 0.85 | 534 207 | 18.38 | −63.99 | 325 765 | 17.16 | 0 | 39.02 |
| 20 × 3 | 441 774 | 1.071 | 441 774 | 1 | 441 774 | 19.11 | 0 | 441 774 | 21.37 | 0 | 0 |
| 20 × 3 | 524 798 | 0.33 | 524 798 | 0.94 | 706 025 | 18.9 | −34.53 | 524 798 | 24.37 | 0 | 25.67 |
| 24 × 3 | 790 815 | 6.81 | 790 815 | 1.1 | 790 815 | 23.58 | 0 | 790 815 | 14.04 | 0 | 0 |
| 25 × 4 | 1 055 866 | 89.82 | 1 055 866 | 1.08 | 1 055 866 | 15.95 | 0 | 1 055 866 | 18.91 | 0 | 0 |
| 25 × 4 | 1 013 013 | 150.67 | 1 013 013 | 1.22 | 1 013 013 | 24 | 0 | 1 013 013 | 23.12 | 0 | 0 |
| 25 × 4 | 1 213 690 | 179.51 | 1 213 690 | 1.13 | 1 215 822 | 20.04 | −0.18 | 1 213 690 | 17.97 | 0 | 0.18 |
| 25 × 4 | 1 189 388 | 128.69 | 1 189 388 | 1.04 | 1 189 388 | 15.78 | 0 | 1 189 388 | 14.7 | 0 | 0 |

Table 3
Results for Test Set 3

| Size ($n \times m$) | ITS | | SA | | | Hybrid SA with TS | | | |
|---|---|---|---|---|---|---|---|---|---|
| | Cost | CPU (s) | Cost | CPU (s) | vs. ITS(%) | Cost | CPU (s) | vs. ITS(%) | vs. SA(%) |
| 100 × 16 | 105 451 | 9.09 | 104 745 | 25.79 | 0.67 | 101 304 | 22.43 | 4.09 | 3.29 |
| 160 × 20 | 180 338 | 22.63 | 186 236 | 36.58 | −3.27 | 176 301 | 58.17 | 2.29 | 5.33 |
| 220 × 24 | 248 361 | 43.96 | 260 805 | 44.55 | −5.01 | 244 302 | 119.45 | 1.66 | 6.33 |
| 280 × 28 | 314 488 | 75.29 | 329 524 | 21.70 | −4.78 | 309 626 | 201.66 | 1.57 | 6.04 |
| 340 × 32 | 382 374 | 119.80 | 404 637 | 26.16 | −5.82 | 378 720 | 298.90 | 0.96 | 6.41 |
| 400 × 36 | 445 238 | 178.22 | 475 777 | 30.62 | −6.86 | 442 172 | 403.92 | 0.69 | 7.06 |
| 460 × 40 | 517 772 | 250.53 | 552 141 | 35.36 | −6.64 | 514 060 | 591.48 | 0.72 | 6.90 |
| 520 × 44 | 582 537 | 347.86 | 623 250 | 39.62 | −7.00 | 579 765 | 741.57 | 0.48 | 6.98 |
| 580 × 48 | 651 299 | 464.28 | 697 060 | 44.99 | −7.03 | 647 446 | 1246.23 | 0.60 | 7.12 |
| 640 × 52 | 724 990 | 587.81 | 764 846 | 49.48 | −5.50 | 712 037 | 1522.75 | 1.82 | 6.90 |

### 8.3. Test Set 3: randomized large input

Hundred test cases are generated in this set, and they are categorized into 10 different sizes with 10 cases in each size. The details are presented in Table 3. From the table, 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 those from ITS. However, we noticed that the CPU time for SA is extremely low. When we attempted to increase the number of iterations, the results were not improved much. We illustrate a case to show the relationship between the objective function cost and number of iterations in Fig. 3, where we see that the objective value drops steeply in the early iterations, but is asymptotic in after this.

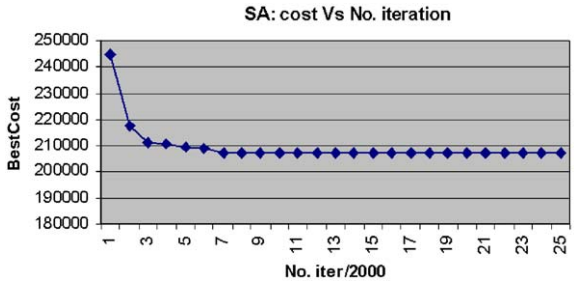


Fig. 3. Relation between bestcost and the number of iterations in SA method.

Table 4
Results for Test Set 4

| Size ($n \times m$) | ITS | | SA | | | Hybrid SA with TS | | | |
|---|---|---|---|---|---|---|---|---|---|
| | Cost | CPU (s) | Cost | CPU (s) | vs. ITS(%) | Cost | CPU (s) | vs. ITS(%) | vs. SA(%) |
| 100 × 16 | 211 500 | 16.675 | 203 469 | 33.23 | 3.80 | 203 030 | 70.27 | 4.17 | 0.22 |
| 140 × 18 | 357 701 | 22.008 | 348 702 | 54.32 | 2.52 | 347 844 | 141.24 | 2.83 | 0.25 |
| 180 × 20 | 496 218 | 36.866 | 488 713 | 66.05 | 1.51 | 487 289 | 191.95 | 1.83 | 0.29 |
| 220 × 22 | 644 356 | 60.48 | 636 556 | 83.68 | 1.21 | 635 684 | 284.50 | 1.36 | 0.14 |
| 260 × 24 | 811 147 | 68.162 | 802 057 | 97.88 | 1.12 | 797 867 | 374.99 | 1.66 | 0.52 |
| 300 × 26 | 1 006 603 | 69.411 | 987 698 | 119.70 | 1.88 | 984 853 | 519.80 | 2.21 | 0.29 |
| 340 × 28 | 1 194 938 | 96.902 | 1 172 399 | 134.22 | 1.89 | 1 168 132 | 613.08 | 2.29 | 0.36 |
| 380 × 30 | 1 401 163 | 130.03 | 1 389 657 | 144.03 | 0.82 | 1 375 722 | 714.72 | 1.85 | 1.00 |
| 420 × 32 | 1 619 008 | 169.448 | 1 600 538 | 154.59 | 1.14 | 1 590 771 | 812.21 | 1.78 | 0.61 |
| 460 × 34 | 1 833 132 | 218.553 | 1 817 486 | 168.64 | 0.85 | 1 802 728 | 920.58 | 1.69 | 0.81 |

### 8.4. 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. Table 4 shows the results for this set.

From the table, we see that hybrid method provides better solutions than SA which performs better than ITS. The hybrid method takes a relatively long running time. Even if we increased the number of iterations in the SA method, we are not likely to get results as good as those from the hybrid method. For large input sizes, the SA algorithm run time is much faster than TS, although it was run on a slower CPU.

### 8.5. 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

Table 5
Results for Test Set 5

| Size ($n \times m$) | ITS | | SA | | | Hybrid SA with TS | | | |
|---|---|---|---|---|---|---|---|---|---|
| | Cost | CPU (s) | Cost | CPU (s) | vs. ITS(%) | Cost | CPU (s) | vs. ITS(%) | vs. SA(%) |
| 100 × 16 | 18 1632 | 8.13 | 177 964 | 27.37 | 2.02 | 175 426 | 35.00 | 3.54 | 1.43 |
| 140 × 18 | 298 082 | 14.14 | 293 401 | 33.14 | 1.57 | 289 373 | 71.43 | 3.01 | 1.37 |
| 180 × 20 | 412 614 | 20.68 | 410 350 | 28.18 | 0.55 | 401 990 | 108.39 | 2.64 | 2.04 |
| 220 × 22 | 533 250 | 27.87 | 533 054 | 36.70 | 0.04 | 521 717 | 162.23 | 2.21 | 2.13 |
| 260 × 24 | 670 608 | 38.66 | 672 229 | 40.20 | −0.24 | 653 386 | 199.98 | 2.64 | 2.80 |
| 300 × 26 | 813 462 | 47.47 | 817 833 | 45.30 | −0.54 | 790 499 | 213.81 | 2.90 | 3.34 |
| 340 × 28 | 948 443 | 83.16 | 966 287 | 50.16 | −1.88 | 931 055 | 298.55 | 1.87 | 3.65 |
| 380 × 30 | 1 122 868 | 111.84 | 1 146 291 | 59.62 | −2.09 | 1 110 928 | 238.44 | 1.07 | 3.08 |
| 420 × 32 | 1 280 326 | 144.77 | 1 318 100 | 60.17 | −2.95 | 1 275 887 | 288.64 | 0.35 | 3.20 |
| 460 × 34 | 1 462 915 | 174.32 | 1 509 599 | 63.47 | −3.19 | 1 468 701 | 221.76 | −0.39 | 2.71 |

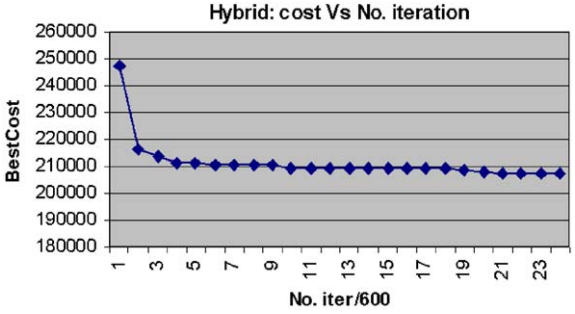


Fig. 4. Relation between bestcost and the number of iterations in hybrid method.

packed input is 100%). These test results are shown in Table 5. From the table, we find that hybrid method gives a relatively large improvement over most of the cases (9 out of 10) when compared with ITS. Additionally, it is better in all of 10 cases than SA.

The relationship, for an example test case, between cost and the number of iterations for hybrid method is illustrated in Fig. 4. From Fig. 3, we see that cost in hybrid method decreases quickly at a low number of iterations, and inherits this advantage of the SA algorithm. However, unlike SA, the cost continues to decrease in later iterations.

We find that for small size input, SA does better than ITS, but for large size input, the ITS algorithm gives better performance. As in Test Set 4, for large input size, SA shows a clearly shorter running time although run on a slower CPU.

From these 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.

## 9. Conclusions and future work

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 the purely SA approach were not as good as the Interval Exchange TS (ITS) approach proposed in [19]. However, the approach has advantages in running time since it was able to reach relatively low values in short times. We took advantage of this property and consequently developed a hybrid method which inserts TS steps into SA, which we found gave better results than both the ITS and SA approaches within reasonable times.

The AGAP is a well-studied problem with many variants and related problems. We expect that these new approaches will be useful in developing solutions for extensions of the over-constrained AGAP. The use of meta-heuristics is relatively new to this problem and may possibly be used for further applications.

We considered cases where flights go into gates as soon as they arrive. If no gate is available at the time they arrive, they are taken to be ungated. It is conceivable that in future research, we can consider the case in which flights are allowed to hold outside gates for a certain period of time before being allocated a gate, a situation which is commonly found in airports.

## References

[1] Braaksma JP, Shortreed JH. Improving airport gate usage with critical path method. Transportation Engineering Journal of ASCE 1971;97(2):187–203.
[2] Obata T. The quadratic assignment problem: evaluation of exact and heuristic algorithms. Technical Report TRS-7901, Rensselaer Polytechnic Institute, Troy, New York, 1979.
[3] Babic O, Teodorovic D, Tosic V. Aircraft stand assignment to minimize walking. Journal of Transportation Engineering 1984;110:55–66.
[4] Mangoubi DFX, Mathaisel RS. Optimizing gate assignments at airport terminals. Transportation Science 1985;19: 173–88.
[5] Bihr R. A conceptual solution to the aircraft gate assignment problem using 0,1 linear programming. Computers and Industrial Engineering 1990;19:280–4.
[6] Wirasinghe SC, Bandara S. Airport gate position estimation for minimum total costs—approximate closed form solution. Transportation Research B 1990;24B(4):287–97.
[7] Yan S, Chang C-M. A network model for gate assignment. Journal of Advanced Transportation 1998;32(2):176–89.
[8] Baron P. A simulation analysis of airport terminal operations. Transportation Research 1969;3:481–91.
[9] Cheng Y. Network-based simulation of aircraft at gates in airport terminals. Journal of Transportation Engineering 1998;124:188–96.
[10] Cheng Y. A rule-based reactive model for the simulation of aircraft on airport gates. Knowledge-Based Systems 1998;10:225–36.
[11] Hassounah MI, Steuart GN. Demand for aircraft gates. Transportation Research 1993;1423:22–33.
[12] Yan S, Huo C-M. Optimization of multiple objective gate assignments. Transportation Research Part A: Policy and Practice 2001;35:413–32.
[13] Zhu Y, Lim A, Rodrigues B. Aircraft and gate scheduling with time windows. In: Proceedings of the 15th IEEE International Conference on Tools with Artificial Intelligence, Sacramento, Califorina, USA, 2003.
[14] Bolat A. Assigning arriving aircraft flights at an airport to available gates. Journal of the Operational Research Society 1999;50:23–34.
[15] Bolat A. Procedures for providing robust gate assignments for arriving aircraft. European Journal of Operations Research 2000;120:63–80.

[16] Gu Y, Chung CA. Genetic algorithm approach to aircraft gate reassignment problem. Journal of Transportation Engineering 1999;125(5):384–9.
[17] Haghani A, Chen MC. Optimizing gate assignments at airport terminals. Transportation Research A 1998;32(6): 437–54.
[18] 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, Island of Maui, Hawaii, USA, 2001.
[19] Ding H, Lim A, Rodrigues B, Zhu Y. Aircraft and gate scheduling optimization at airports. In: Proceedings of the 37th Hawaii International Conference on System Sciences, Big Island, Hawaii, USA, 2003.
[20] Kirkpatrick S, Gellatt CD, Vecchi MP. Optimization by simulated annealing. Science 1993;220(4598):671–80.