The Airport Gate Assignment Problem: Mathematical Model and a Tabu Search Algorithm

Jiefeng Xu and Glenn Bailey
Delta Technology Inc., 1001 International Boulevard,
Atlanta, Georgia 30354-1801
Email: jiefeng.xu@delta-air.com, glenn.bailey@delta-air.com

Abstract

In this paper, we consider an Airport Gate Assignment Problem that dynamically assigns airport gates to scheduled flights based on passengers' daily origin and destination flow data. The objective of the problem is to minimize the overall connection times that passengers walk to catch their connection flights. We formulate this problem as a mixed 0-1 quadratic integer programming problem and then reformulate it as a mixed 0-1 integer problem with a linear objective function and constraints. We design a simple tabu search meta-heuristic to solve the problem. The algorithm exploits the special properties of different types of neighborhood moves, and create highly effective candidate list strategies. We also address issues of tabu short term memory, dynamic tabu tenure, aspiration rule, and various intensification and diversification strategies. Preliminary computational experiments are conducted and the results are presented and analyzed.

1 Introduction

The airline industry has long been a fertile area for applying optimization techniques. In this paper, we consider an optimization problem that allows an airline company to dynamically assign existing airport gates to its scheduled flights based on passengers' daily origin and destination (O&D) data. The objective of this optimization is to minimize the overall connection times required for passengers to catch their connection flights. The Airport Gate Assignment Problem (AGAP) can be described in detail as follows. Suppose an airline company owns/leases a certain number of gates in an airport, and hosts a certain number of flights every day. Currently, assigning the flights to gates is static and it does not consider passenger connection times between flights. The AGAP is a planning problem that can be solved at the beginning of every planning day. Since most passengers' O&D data are available at that moment, the AGAP can determine the feasible gate-flight assignment such that the total of passengers' connection times is minimized. For a passenger who transfers to a connection flight at the airport, the connection time is readily defined as the walking time required from the arrival gate of his/her incoming flight to the departure gate of his/her outgoing flight. For a passenger whose destination is the current airport, the connection time is defined as the walking time required from his/her arrival gate to the baggage claim area. For a passenger who originates at the current airport, the connection time is accordingly defined as the walking time required from the baggage check-in area to the departure gate of his/her outgoing flight. In the AGAP, the arrival time of the incoming flight and the departure time of the outgoing flight are fixed due to the published flight schedule. Each flight must be assigned to exactly one gate, and there should be sufficient time for passengers boarding at the gate.

In addition to the increased customer service level by minimizing the connection time, the AGAP may achieve another associated benefit for the company: the total routing costs to transfer passengers' luggage may be minimized via the optimization of the gate-flight assignment. Despite these benefits in practice, the AGAP received little attention from the operations research community. To our best knowledge, there exists no prior research directly addressing the AGAP to minimize the customer connection time.

This paper is organized as follows. We present the mathematical formulation in the next section. We also discuss the linearization of the quadratic model. In Section 3 we describe a Tabu Search (TS) based heuristic for the problem and examine several relevant issues such as short term and long term memory, move selection, neighborhood structure, and candidate list strategies. Section 4 reports computational results from a set of carefully designed test problems, including compari-

son with an exact approach. In the concluding section, we summarize our methodology and findings.

2 Mathematical Formulation

In this section, we establish the mathematical models for the AGAP. First we describe the following input data required by the AGAP:

- N : set of flights (arrived at or) departed from the airport during the planning day;
- K: set of gates available at the airport;
- a_i : arrival time of flight i;
- d_i : departure time of flight i;
- c_{kl} : connection time for passengers from arrival gate k to departure gate l;
- f_{ij}: the number of passengers transferring from arrival flight i to departure flight j;
- θ_i : average boarding time per passenger for flight i.
- α : buffer time between the aircraft's arrival time and the start time for passenger boarding.
- β : buffer time between the aircraft's departure time and the next aircraft's arrival time at the same gate.
- M: a sufficiently large number.

Some special notes need to be made for the input data. First, the gate set K contains a dummy gate (denoted as 0) which represents the entrance/exit areas of the airport. Therefore, c_{0i} represents the connection time from the airport entrance (or baggage check-in counters) to departure gate i, and c_{i0} is the connection time from gate i to the airport exit (or baggage claim) area. Similarly, the flight set N includes a dummy flight denoted as 0, which indicates the airport itself. Therefore, f_{i0} represents the number of passengers from arrival flight i whose destination is the current airport, and f_{0i} is the number of passengers who originates from the current airport and take departure flight i. Accordingly, a_0 is designated as the start time, d_0 is the end time of the planning day, and θ_0 is zero.

Second, for each departure flight i there is an associated incoming flight which arrives at the same gate immediately before flight i departs, with the assumption that these two flights use the same aircraft. Though the arrival flight may carry a different flight number, we still denote it as flight i in order to simplify the model and to avoid an unnecessary proliferation of symbols and subscripts. Thus, N only represents half of the number of flights, with each flight in N denoting both an incoming flight and an outgoing flight. Therefore, a_i stands for the arrival time of the arrival flight i, and d_i indicates the departure time for the departure flight. Both incoming and outgoing flights are assigned to the same gate and use the same aircraft. In this regard, f_{ij} represents the flow from the incoming flight i to the outgoing flight j. Furthermore, $\sum_{i\in N} f_{ij}$ represent the total number of passengers who will be on-broad for flight j.

Finally, the two buffer times make sense in practice. The first buffer time α, can be used for unboarding passengers and crew members from the incoming flight, for cleaning the cabin and supplementing the supplies, and for boarding the crew members of the departure flight. The second buffer time, β, can be used for ground clearance to receive the next aircraft's arrival.

We illustrate the AGAP using a mathematical formulation with the following decision variables:

- y_{ik} : a binary variable equal to 1 if and only if flight i is assigned to gate k;
- z_{ijk} : a binary variable equal to 1 if and only if both flights i and j are assigned to gate k and flight i immediately precedes flight j;
- t_i : time the gate starts to open for boarding for flight i.

The mathematical formulation is as follows:

Model 1

$$\textbf{Minimize} \sum_{i,j\in N}\sum_{k,l\in K} f_{ij}c_{kl}y_{ik}y_{jl} \tag{1}$$

subject to:

$$\sum_{k\in K} y_{ik} = 1, \quad \forall i \in N, \tag{2}$$

$$y_{ik} = \sum_{j\in N} z_{ijk}, \quad \forall i \in N, \forall k \in K, \tag{3}$$

$$y_{jk} = \sum_{i\in N} z_{ijk}, \quad \forall j \in N, \forall k \in K, \tag{4}$$

$$t_i \geq a_i + \alpha, \quad \forall i \in N, \tag{5}$$

$$t_i \leq d_i - \theta_i \sum_{j\in N} f_{ji}, \quad \forall i \in N, \tag{6}$$

$$t_i + \theta_i \sum_{j\in N} f_{ji} \leq t_j + (1 - z_{ijk})M, \quad \forall i, j \in N, \forall k \in K, \tag{7}$$

$$a_j + (1 - z_{ijk})M \geq d_i + \beta, \quad \forall i, j \in N, \forall k \in K, \tag{8}$$

$$y_{ik}, z_{ijk} \in \{0, 1\}, \quad \forall i, j \in N, \forall k, l \in K, \tag{9}$$

$$t_i \geq 0 \quad \forall i \in N. \tag{10}$$

In this formulation, the objective function (1) seeks to minimize the total connection times by passengers.

Constraint (2) specifies that every flight must be assigned to one gate. Constraint (3) indicates that every flight can have at most one flight immediately followed at the same gate. Constraint (4) indicates that every flight can have at most one preceding flight at the same gate. Constraints (5) and (6) stipulate a gate must open for boarding on a flight during the time between its arrival and departure, and also must allow sufficient time for handling the passenger boarding, which is assumed to be proportional to the number of passengers going on board. Constraint (7) establishes the precedence relationship for the binary variable z_{ijk} and the time variables t_i and t_j, and is only effective when $z_{ijk} = 1$. It demands that if flight i is assigned immediately before flight j at the same gate k, the gate must open for flight i earlier than for flight j. Therefore, it ensures each gate only serves one flight at any particular time. Constraint (8) further states the aircraft can only arrive at the gate when the previous flight has departed for certain time. Note that (5)–(8) jointly imply that $a_i + \alpha + \theta_i \sum_{j \in N} f_{ji} \leq d_i$, which reveals a relationship between the input data. Finally, we define the binary and nonnegative requirements for decision variables in constraints (9) and (10).

The above-mentioned model is a mixed 0-1 integer programming model with the quadratic objective function. We use a common approach to reformulate the model into a mixed 0-1 integer problem with a linear objective function and constraints (See [2],[6] and [13]). First we introduce the binary variable x_{ijkl} to replace the binary term $y_{ik}y_{jl}$, so $x_{ijkl} = 1$ if and only if flight i is assigned to gate k and flight j is assigned to gate l. Therefore the Model 1 can be reformulated as follows.

Model 2

$$\textbf{Minimize} \sum_{i,j \in N} \sum_{k,l \in K} f_{ij} c_{kl} x_{ijkl} \tag{11}$$

subject to:

$$(2) - (10)$$

$$x_{ijkl} \leq y_{ik}, \quad \forall i,j \in N, \ \forall k,l \in K \tag{12}$$

$$x_{ijkl} \leq y_{jl}, \quad \forall i,j \in N, \ \forall k,l \in K \tag{13}$$

$$y_{ik} + y_{jl} - 1 \leq x_{ijkl}, \quad \forall i,j \in N, \ \forall k,l \in K, \tag{14}$$

$$x_{ijkl} \in \{0,1\}, \quad \forall i,j \in N, \ \forall k,l \in K, \tag{15}$$

Constraints (12) and (13) state that a binary variable x_{ijkl} can be equal to one if flight i is assigned to gate k ($y_{ik} = 1$) and flight j is assigned to gate l ($y_{jl} = 1$). Constraint (14) further specifies the necessary condition that x_{ijkl} must be equal to one if $y_{ik} = 1$ and $y_{jl} = 1$.

Both Models 1 and Model 2 are NP-hard, which implies that there is no known algorithm for finding the optimal solution within a polynomial-bounded amount of time. In practice, a major airline hub may handle more than 750 daily flights at more than 100 gates, which in our formulation results in billions of binary variables. Due to such a huge size, this model cannot be handled by branch-and-bound-based MIP solvers within a reasonable time bound. Thus, the design of an efficient heuristic becomes of paramount importance and constitutes the key focus of this important application.

3 A Tabu Search Algorithm

In this section we present a simple TS algorithm which is specially designed for the AGAP. TS is a metaheuristic approach that is recognized as a very effective tool for many combinatorial optimization problems. The basic TS approach is to search for the optimum solution with the assistance of an adaptive memory procedure that proceeds as follows. At each iteration, candidate neighborhood moves are evaluated, which then lead from the current solution to a new solution. Restrictions are imposed to classify certain moves tabu and thus forbid (or discourage) their selection. Conventionally, a non-tabu move with the highest evaluation is selected, although aspiration criteria permit sufficiently attractive moves to be selected in spite of their tabu status. The algorithm terminates after a pre-defined number of iterations. A complete review on TS can be found in [4]and [5].

The elementary TS form consists of a neighborhood search and the use of recency-based short-term memory. We describe these components as well as enhancements in the next several subsections.

3.1 Neighborhood Search Moves

A neighborhood search move is the operation that maps one solution π to another solution π'. All solutions that are "reachable" from solution π via a single move form a neighborhood of π, denoted by $N(\pi)$. The attractiveness of a move from π to π' can be examined by calculating the cost (objective function value) improvement of π' compared to that of π. Here we develop three types of neighborhood moves that prevail in the AGAP applications.

The first type of neighborhood move is called as *Insert Move*. It moves a single flight to a gate other than the one it currently assigns. We denote $(i,k) \rightarrow (i,l)$ as the insert which inserts flight i from gate k to gate

l. We denote the current solution as π and its corresponding objective function value as $f(\pi)$. The resulting solution of the move $(i,k) \rightarrow (i,l)$ is denoted as π'. Therefore, the cost improvement of the *Insert Move* $(i,k) \rightarrow l$ (denoted as $\Delta_{(i,k)\rightarrow(i,l)}$) can be calculated as follows:

$$\begin{aligned}
&\Delta_{(i,k)\rightarrow(i,l)} = f(\pi') - f(\pi) \\
&= \sum_{p\neq i, q\neq i} \sum_{r,s\in K} c_{rs} f_{pq} y'_{pr} y'_{qs} \\
&+ \sum_{p\neq i} \sum_{r,s\in K} c_{rs} f_{pi} y'_{pr} y'_{is} + \sum_{q\neq i} \sum_{r,s\in K} c_{rs} f_{iq} y'_{ir} y'_{qs} \\
&+ \sum_{r,s\in K} c_{rs} f_{ii} y'_{ir} y'_{is} - \sum_{p\neq i, q\neq i} \sum_{r,s\in K} c_{rs} f_{pq} y_{pr} y_{qs} \\
&- \sum_{p\neq i} \sum_{r,s\in K} c_{rs} f_{pi} y_{pr} y_{is} - \sum_{q\neq i} \sum_{r,s\in K} c_{rs} f_{iq} y_{ir} \\
&- \sum_{r,s\in K} c_{rs} f_{ii} y_{ir} y_{is} \qquad (16)
\end{aligned}$$

Since by assumption we have $y_{ik} = 1, y_{ir} = 0 \ \ \forall r \neq k$, $y'_{il} = 1, y'_{ir} = 0 \ \ \forall r \neq l$ and $y_{pr} = y'_{pr} \ \ \ \forall p \neq i$, $f_{ii} = 0$, and $c_{kl} = c_{lk}$, then (16) becomes:

$$\begin{aligned}
\Delta_{(i,k)\rightarrow(i,l)} &= \textstyle\sum_{p\neq i} \sum_{r\in K} (c_{rl} - c_{rk}) f_{pi} y_{pr} \\
&+ \sum_{q\neq i} \sum_{s\in K} (c_{ls} - c_{ks}) f_{iq} y_{qs} \\
&= \textstyle\sum_{p\neq i} \sum_{r\in K} (c_{rl} - c_{rk})(f_{pi} + f_{ip}) y_{pr} \qquad (17)
\end{aligned}$$

In reality, for two distinct flights i and j, f_{ij} and f_{ji} are exclusive, that is, if $f_{ij} > 0$ then $f_{ji} = 0$, and vice versa. Thus, denoting $F_{ij} = \max\{f_{ij}, f_{ji}\}$, (17) can be simplified as

$$\Delta_{(i,k)\rightarrow(i,l)} = \sum_{p\neq i} \sum_{r\in K} (c_{rl} - c_{rk}) F_{pi} y_{pr}. \qquad (18)$$

Since $(c_{rl} - c_{rk}) F_{pi}$ is independent of the gate assignment and can be pre-calculated, so an *Insert Move* can be evaluated in linear time at each iteration.

The second type of neighborhood move, which is denoted as *Exchange I Move*, is to exchange two flights and their gate assignment. *Exchange I Move* represents a more complicated move. For *Exchange I Move*, we have the following observations:

Observation 1: The Exchange I Move is infeasible for the same-gate flights.

For proof we assume there exists subscripts i, j, k such that $z_{ijk} = 1$. By (5) and (6), we have $d_i \geq t_i + \theta \sum_{j\in N} f_{ji} \geq a_i + \alpha > a_i$. By (8), we have $a_j \geq d_i + \beta > a_i$. Therefore, i and j is not exchangable.

Observation 2: An Exchange I Move can be viewed as two Insert moves.

To illustrate this, we evaluate the cost improvement of the *Exchange I move*. We denote the current solution as π and its corresponding objective function value as $f(\pi)$. In π, we assume there exist two distinct flights i and j, and two distinct gates k and l, such that $y_{ik} = 1$ and $y_{jl} = 1$. We consider the *Exchange I Move* for $(i,k) \Longleftrightarrow (j,l)$. We define the space $H = p \in N; r \in K$ and the space $A = \{p = i; r = k\} \cup \{p = i; r = l\} \cup \{p = j; r = l\} \cup \{p = j; r = k\}$. We denote that $\bar{A} = H - A$. The new solution after $(i,k) \Longleftrightarrow (j,l)$, π', can be constituted as follows:

$$y'_{pr} = \begin{cases} y_{pr} & (p,r) \in \bar{A}; \\ 1 - y_{pr} & (p,r) \in A. \end{cases} \qquad (19)$$

Then the cost improvement for the move $(i,k) \Longleftrightarrow (j,l)$ can be calculated as

$$\begin{aligned}
&\Delta_{(i,k)\Longleftrightarrow(j,l)} = f(\pi') - f(\pi) \\
&= \sum_{p,r\in A} \sum_{q,s\in A} c_{rs} f_{pq} - \sum_{p,r\in A} \sum_{q,s\in A} c_{rs} f_{pq} y_{pr} \\
&+ \sum_{p,r\in \bar{A}} \sum_{q,s\in A} c_{rs} f_{pq} y_{pr} - \sum_{p,r\in A} \sum_{q,s\in A} c_{rs} f_{pq} y_{qs} \\
&-2 \sum_{p,r\in \bar{A}} \sum_{q,s\in A} c_{rs} f_{pq} y_{pr} y_{qs} + \sum_{p,r\in A} \sum_{q,s\in \bar{A}} c_{rs} f_{pq} y_{qs} \\
&-2 \sum_{p,r\in A} \sum_{q,s\in \bar{A}} c_{rs} f_{pq} y_{pr} y_{qs} \qquad (20)
\end{aligned}$$

Noting that $y_{il} = 0, \ \ y_{ik} = 1, \ \ y_{jk} = 0, \ y_{jl} = 1, \ c_{kl} = c_{lk}, \ \ c_{kk} = c_{ll} = 0, \ \ f_{ii} = f_{jj} = 0$, applying them to (20) gives

$$\begin{aligned}
\Delta_{(i,k)\Longleftrightarrow(j,l)} &= \sum_{p,r\in \bar{A}} (c_{rl} - c_{rk})(f_{pi} + f_{ip} - f_{pj} - f_{jp}) y_{pr} \\
&= \sum_{p,r\in \bar{A}} (c_{rl} - c_{rk})(F_{pi} - F_{pj}) y_{pr} \\
&= \Delta_{(i,k)\rightarrow(i,l)} + \Delta_{(j,l)\rightarrow(j,k)}. \qquad (21)
\end{aligned}$$

Clearly, the move $(i,k) \Longleftrightarrow (j,l)$ can be viewed as a composition of two *Insert Moves* $(i,k) \rightarrow (i,l)$ and $(j,l) \rightarrow (j,k)$. Therefore, an *Exchange I Move* can also be evaluated in linear time at each iteration.

The third type of neighborhood move, denoted as *Exchange II Move*, exchanges two flight pairs in the current assignment. More specifically, assuming there exists two triples (i_1, i_2, k) and (j_1, j_2, l), such that $z_{i_1 i_2 k} = 1$ and $z_{j_1 j_2 l} = 1$, the move exchanges the flight pair (i_1, i_2) to gate l, and the flight pair (j_1, j_2) to gate k. We denote the move as $(i_1, i_2, k) \Longleftrightarrow (j_1, j_2, l)$.

Similarly, we denote $A^2 = \{p = i_1; r = k\} \cup \{p = i_1; r = l\} \cup \{p = i_2; r = k\} \cup \{p = i_1; r = l\} \cup \{p = j_1; r = l\} \cup \{p = j_1; r = k\} \cup \{p = j_2; r = l\} \cup \{p = j_2; r = k\}$. Again, we denote that $\bar{A^2} = H - A^2$. The new solution after $(i_1, i_2, k) \Longleftrightarrow (j_1, j_2, l)$, π', can be constituted as

$$y'_{pr} = \begin{cases} y_{pr} & (p,r) \in \bar{A^2}; \\ 1 - y_{pr} & (p,r) \in A^2. \end{cases} \quad (22)$$

Using a similar approach applied to (20), we conclude that the cost improvement of *Exchange II Move* can be evaluated linearly. More specifically, we have

$$\begin{aligned} &\Delta_{(i_1,i_2,k)\Longleftrightarrow(j_1,j_2,l)} \\ &= \sum_{p,r \in \bar{A^2}} (c_{rk} - c_{rl})(F_{pj_1} + F_{pj_2} - F_{pi_1} - F_{pi_2}) y_{pr} \\ &= \Delta_{(i_1,k)\Longleftrightarrow(j_1,l)} + \Delta_{(i_2,k)\Longleftrightarrow(j_2,l)} \end{aligned} \quad (23)$$

Similarly, an *Exchange II Move* can be viewed as a composition of two *Exchange I* moves, and its evaluation can be acquired by the evaluations of its two partial *Exchange I* moves.

3.2 Tabu Short-Term Memory

The elementary tabu search procedure innovatively incorporates the search history (memory) by imposing *tabu restrictions* to prevent revisiting previous solutions, thus avoiding being trapped into local optima. More specifically, it forbids the solution attribute changes recorded in the recency-memory to be reversed back to their previous settings. Tabu restrictions often operate through short term memory, which records the most recent changes of selected solution attributes. By short term memory, a certain TS restriction can only be valid for a limited time, and is referred to as *tabu tenure* for a move. Therefore, once a move is made we forbid those future moves which would reverse the attribute changes of that move for t iterations, where t is the value of the tabu tenure determined right after the move is made.

For the AGAP, the short term memory can be efficiently implemented as follows. We maintain the three lists $tabu(), tabuI()$ and $tabuII()$ for the three corresponding moves, and denote $tabu((i,k) \rightarrow (i,l))$, $tabuI((i,k) \Longleftrightarrow (j,l))$ and $tabuII((i_1,i_2,k) \Longleftrightarrow (j_1,j_2,l))$ to indicate the future iteration values governing the duration that will forbid the reversal of the expressed *Insert*, *Exchange I*, or *Exchange II* moves. Initially, all values of the elements in these three lists are set to zero. Letting *iter* be the current iteration counter, then the TS restrictions are:

$tabu((i,k) \rightarrow (i,l)) = iter + tabu\_tenure$ (for the *Insert* move of $(i,k) \rightarrow (i,l)$, to prevent the move of $(i,l) \rightarrow (i,k)$),
$tabuI((i,k) \Longleftrightarrow (j,l)) = iter + tabu\_tenure$ (for the *ExchangeI* move of $(i,k) \Longleftrightarrow (j,l)$, to prevent the move of $(j,k) \Longleftrightarrow (i,l)$),
$tabuII((i_1,i_2,k) \Longleftrightarrow (j_1,j_2,l)) = iter + tabu\_tenure$ (for the *ExchangeII* move of $(i_1,i_2,k) \Longleftrightarrow (j_1,j_2,l)$, to prevent the move of $(i_1,i_2,l) \Longleftrightarrow (j_1,j_2,k)$).

As a result, it is easy to test if the TS restrictions are in effect. The TS restrictions are valid if and only if $iter \leq tabu((i,k) \rightarrow (i,l))$ for the candidate *Insert* move of $(i,k) \rightarrow (i,l)$, or $iter \leq tabuI((i,k) \Longleftrightarrow (j,l))$ for the candidate *Exchange I* move of $(i,k) \Longleftrightarrow (j,l)$, or $iter \leq tabuII((i_1,i_2,k) \Longleftrightarrow (j_1,j_2,l))$ for candidate *Exchange II* move of $(i_1,i_2,k) \Longleftrightarrow (j_1,j_2,l))$.

We employ the dynamic rule to select the value of tabu tenure at each iteration uniformly within a pre-defined interval. This helps to significantly diminish the possibility of cycling during the search. We also use different tabu tenure intervals for different neighborhood moves. For example, the values of *tabu_tenure* used for $tabuI((i,k) \Longleftrightarrow (j,l))$ and $tabuII((i_1,i_2,k) \Longleftrightarrow (j_1,j_2,l))$ could be different. Although there exists a more complicated option called *reactive tabu search* to systematically alter the value of tabu tenure as proposed by [1], we adhere to the dynamic tabu tenure for its simplicity and effectiveness demonstrated in various TS applications (See [7],[8],[9],[11] and [12]).

In TS applications, the tabu restrictions can be overridden if the candidate move under consideration is "good enough". The rule to determine whether to ignore the tabu restriction is called *aspiration criterion*. In the AGAP, we apply the literature's standard aspiration criterion, which is to remove the TS restrictions of a move if the move would lead to a solution better than the best solution obtained so far.

Below we outline a general framework of the short term memory based tabu search heuristic for the AGAP. Let *max_iter* be the maximum iteration number allowed for the search. All other notations remain the same as previously defined.

Step 0 (Initialization).
Generate a starting solution π_now; set the current best known solution π_best as $\pi\_best = \pi\_now$. Initialize all *tabu*, *tabuI* and *tabuII* memory as 0. Set $ietr = 0$.

Step 1 (Termination Conditions).
If $iter \geq max\_iter$, terminate with π_best. Otherwise, go to Step 2.

Step 2 (Move Selection).

Determine the type of neighborhood move used at the current iteration. Generate the specified neighborhood set of π_now, $N(\pi\_now)$. For each candidate solution $\pi\_trial \in N(\pi\_now)$, evaluate $f(\pi\_trial)$. If $\exists \pi\_trial$ such that $f(\pi\_trial) < f(\pi\_best)$, then select it as $\pi\_next = \pi\_trial$. Otherwise, select $\pi\_next = \min_{\pi\_trial \in N(\pi\_now)} f(\pi\_trial)$ such that $iter \geq tabu(\pi\_now \rightarrow \pi\_trial)$ if the move type is *Insert*, or $iter \geq tabuI(\pi\_now \rightarrow \pi\_trial)$ if the move type is *Exchange I*, or $iter \geq tabuII(\pi\_now \rightarrow \pi\_trial)$ if the move type is *Exchange II*.

Step 3 (Update).

Update the tabu memory as follows:
$tabu(\pi\_now \rightarrow \pi\_next) = iter + U(a_1, b_1)$ for move type *Insert*.
$tabuI(\pi\_now \rightarrow \pi\_next) = iter + U(a_2, b_2)$ for move type *Exchange I*.
$tabuII(\pi\_now \rightarrow \pi\_next) = iter + U(a_3, b_3)$ for move type *Exchange II*.

where $U(a, b)$ represents a number generated randomly between the two bounds a and b. a_1 and b_1 are pre-defined bounds for *Insert* moves, a_2 and b_2 are bounds for *Exchange I* moves, and a_3 and b_3 are bounds for *Exchange II* moves. Then we update $\pi\_now = \pi\_next$. If $f(\pi\_now) < f(\pi\_best)$, then $\pi\_best = \pi\_now$. Finally, we increase iteration counter as $iter = iter + 1$ and go back to Step 1.

As pointed out by many researchers, the simple elementary Tabu Search often produces high quality solutions for a range of combinatorial optimization problems. However, it is now widely recognized that the simple TS algorithm can be effectively enhanced by including advanced strategies such as candidate list strategy, and intensification and diversification strategy. Next we describe these components.

3.3 Intensification and Diversification Strategies

Intensification and diversification may often improve the power of a meta-heuristic search method. Intensification will search one specific region in the solution space more thoroughly, while diversification guides the search to some regions which have received less attention in the search history. In this paper, we develop a simple diversification strategy based on an oscillation mechanism between different neighborhood moves.

As described in 3.1, we employ three different types of neighborhood moves. The first type of move, the *Insert* move, will change single flight's gate assignment. The *Exchange I* move will exchange two flights' gate assignments, and the *Exchange II* move will bring even more changes to the current assignment. Therefore, these moves possess different diversification impact on the search process. More specifically, executing an *Exchange II* move diversifies the search more than executing an *Exchange I* move, and executing an *Exchange I* move makes more diversification effect than an *Insert* move.

Due to these various diversification impacts by our neighborhood moves, we design an oscillation mechanism to guide the search to achieve more diversification under certain situations. First, we apply the different neighborhood moves at different frequency. We execute the *Insert* moves more often than the other two moves to take advantage of its intensification characteristics. The *Exchange I* and *Exchange II* moves are executed less frequently on a periodical basis. Therefore, they may break potential impasse by directing the search to more diversified regions. Next, we execute *Exchange I* and *Exchange II* move for some certain situations which require more diversification. For example, if the current solution cannot make a feasible *Insert* move, or the search shows no improvement for a specified period of time, we will execute an *Exchange I* move or *Exchange II* move.

3.4 Candidate List Strategies

Candidate list strategies prevail in most TS applications and are critical to the efficiency of TS algorithms. The purpose of the candidate list is to isolate a subset of promising neighborhood moves (called candidates) for evaluation. The move is selected within these candidate evaluations. Candidate list strategies can also produce an intensification effect that could improve the overall performance of TS. In this application, we implement candidate list strategy based on the feasibility measure of the neighborhood moves.

Consider an *Insert* move $(i, k) \rightarrow (i, l)$, and assume that flight i will be inserted between flight p and q at gate l. By (5)–(8), the necessary conditions for this move are $a_i > d_p + \beta$ and $d_i < a_q - \beta$. Based on these conditions, we can build a candidate list by enlisting the gates which can accept the *Insert* move with flight i. After a move, most candidate lists remain unchanged, but those who are involved with the affected gate-flight assignment are altered by that move. Similarly, can-

didate list strategies can also be applied to *Exchange I* and *Exchange II* moves by including pre-determined feasible moves in the list and updating the partial list after a move is selected and executed.

The feasibility constraints of the neighborhood moves may be relaxed to a moderate degree to allow moves which lead to infeasible solutions. This is based on the conjecture that there exists infeasible solution regions between feasible solution regions in the search space. Though currently not implemented, our candidate list strategy can be readily adapted to this controlled feasibility violation and therefore provide a bridge for traversing from one feasible solution region to the other.

Furthermore, our candidate list strategy is implicitly implemented to avoid evaluating moves that were previously evaluated. For example, for each type of neighborhood moves, since only a portion of the gate-flight configuration has been changed for each iteration, we can save the move evaluations that are not affected in the last iteration and use the evaluations from the current iteration. This implicit candidate list strategy significantly increases the efficiency of our algorithm.

4 Computational Results

We conduct preliminary computational experiments to verify and validate our AGAP model and TS heuristic. In the next few subsections, we describe the test problem design and parameter settings for our TS heuristics. We report computational results on these test problems and provide analysis.

4.1 Test Problem Design

We designed two sets of test problems. The first set of test problems provides a test platform to validate the objective of the AGAP model. To simulate the current static gate assignment operation, we generate a set of feasible gate assignments with various sizes of flights and gates. We then randomly generate the passengers' O&D data for a certain number of work days for each known gate assignment. We first calculate the total passengers' connection times (1) by the current static gate assignment. Then, for each working day we solve the corresponding AGAP problem using our TS heuristic, and calculate the same objective function value over the new assignment. Finally, we compare the results from both the static assignment and the dynamic optimization model to validate the economic significance of our dynamic AGAP model.

More specifically, the first set of test problems are generated as follows. We generate 7 test problems with up to 50 gates. For each gate we assign 8 flights. The $(n \times m)$ column in Table 1 lists the size for each problem where n and m represent the number of flights and the number of gates respectively. The arrival time of flight i, a_i, is randomly generated within the interval $[90 * (k-1) - 5, 90 * (k-1) + 5]$, where flight i is the kth flight assigned to the gate during the day. We then set the departure time d_i to $a_i + U(55, 65)$. The two buffer times, α and β, are set to 15 and 5 respectively. To better simulate the gate connection time in practice, we arrange the gates equally into two parallel terminals whose layout is illustrated in Figure 1. The connection time from one gate to its adjacent gate on the same terminal is 1 minute, and it takes 3 minutes to walk between the two closest gates at the different terminals, e.g., gate 1 of terminal A and gate 1 of terminal B. Furthermore, the airport entrance/exit is 5 minutes apart from the closet gates (Gate 1) of both terminals.

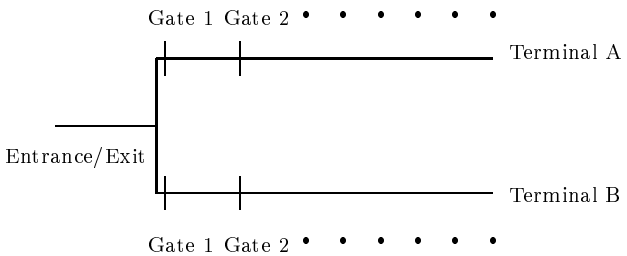

Figure 1: Gate Layout for First Type of Test Problems

We generate the passenger's O&D data as follows. The passenger flow from flight i to flight j, f_{ij}, is generated between the interval [1,10] if $a_i + \alpha < d_j$, and f_{ij} is set to 0 if $a_i + \alpha >= d_j$. The number of passengers for flight i whose origins are the current airport, f_{0i}, is uniformly generated from $[6, 60]$. Similarly, the number of passengers from flight i whose destinations are the current airport, is uniformly generated from the same interval $[6, 60]$. The average boarding time per passenger, θ_i, is randomly generated from the interval $[0.5, 1.0]$. After all these data are generated, we check the consistency of the input data. If the condition $a_i + \alpha + \theta_i * \sum_{j \in N} f_{ji} <= d_i$ is violated, we reduce the value of θ_i to $(d_i - 5 - a_i - \alpha) / \sum_{j \in N} f_{ji}$. For each test problem, we generate the passenger's O&D data for five consecutive working days so that we can validate the benefits of our dynamic assignment model over the static assignment model.

We devise the second set of test problems to show the performance of our TS heuristic. Since our AGAP study is the first known research on this problem and there are no other algorithms available, we compare our heuristic with exact methods. Since the model

contains a huge number of binary variables, the size of the instances that can be solved by exact methods is very limited. Subsequently we show that a MIP-based exact method can only solve the AGAP with up to 20 flights and 5 gates, which is trivial for our TS heuristic.

We generate 10 instances for the second set of test problems. The sizes of these instances are specified in the $(n \times m)$ column in Table 2. Unlike the first set of test problems which start with a feasible gate assignment, we generate the flights' arrival and departure time randomly. First we generate $a_i = U(0, 900)$, then we set $d_i = a_i + U(45, 60)$. The passengers' O&D data, f_{ij}, are generated in the same way as in the first test set, but we only produce data for one working day for each problem. Again, θ_i is generated and adjusted as described in the first test set. To calculate the connection time between two gates, we first obtain the two coordinates $x_i = U(0, 10)$ and $y_i = U(0, 2)$ for every gate i, then we calculate $c_{ij} = |x_i - x_j| + |y_i| + |y_j|$ where $|a|$ represents the absolute value of number a. Since this set of test problems are generated more randomly, some instances may not have feasible solutions. We abandon those infeasible problems identified by the exact method (as described subsequently), and re-generate the problems until feasible solutions are found.

We realize that the computational experiments we currently undertake are merely preliminary and by no means sufficient. Additional simulated data with larger sizes as well as actual operational data from real world are necessary to further test our heuristic. This will become one of the focuses of our future research on the AGAP.

4.2 Parameter Settings for TS Heuristic

Our TS heuristic requires a few parameters which are described subsequently. Note that the parameters of our TS approach are set arbitrarily or by common sense in this paper. This does indicate the potentials for future improvement of our TS heuristic by systematically fine-tuning these parameters using statistical tests, as shown in [10].

Our TS heuristic starts with iteration 1 and terminates at the iteration number which equals $40m - 500$ where m represents the number of gates. The tabu tenure is uniformly generated within the interval [2,5] for *Insert* moves, and [3,7] for *Exchange I* moves, and [4,8] for *Exchange II* moves. Different neighborhood moves are evaluated and executed at different frequency. The *Insert* move is the primary type of moves. It is evaluated and executed at each iteration unless the following conditions specified for the other two neighborhood moves are satisfied. If the iteration counter hits the multiple of seven, e.g., 7, 14, 21, ..., we then evaluate and execute *Exchange II* moves at the current iteration. If the iteration counter hits the multiple of five (e.g., 5, 10, ...), we then evaluate and execute *Exchange I* moves at the current iteration. In addition, if an *Insert* move evaluation does not produce a legitimate (*Insert*) move, we switch to an *Exchange I* move at the next iteration. Similarly, if an *Exchange I* move evaluation does not produce a legitimate (*Exchange I*) move, we switch to an *Exchange II* move at the next iteration. If the number of iterations that the search does not improve the best solution exceeds 50, we switch to *Exchange I* or *Exchange II* moves arbitrarily for the current iteration.

4.3 Results and Analysis

We code our TS heuristic in C++ and compile it with aCC (ansi C++) compiler on an HP D380 workstation. We present our computational results on the two sets of test problems as follows. First, we show the results from the first set of test problems in Table 1.

Problem ($n \times m$)	Static (min.)	TS Savings			CPU (sec.)
		Max (%)	Min (%)	Total (%)	
160 × 20	3180239	17.13	12.32	14.21	2.9
200 × 25	5756391	26.47	19.34	21.40	8.0
240 × 30	9471432	27.89	22.71	24.99	18.98
280 × 35	14510099	29.53	26.24	27.57	35.86
320 × 40	20955371	29.35	26.06	27.67	65.064
360 × 45	29247456	30.52	23.56	27.77	102.3
400 × 50	39362349	31.10	26.50	29.29	154.81
Average	—	—	—	24.70	

Table 1: Results on First Set of Test Problems

In Table 1, we list the problem dimension in $n \times m$ column. The total passengers' connection times over the five consecutive working days under the current static assignment operation are presented in *Static* column. To show the savings achieved by our TS heuristic, we in turn list the maximum, minimum and total saving percentages among the five working days. Finally, we report the average CPU time required for our heuristic to solve a daily AGAP problem with the specified size. Table 1 demonstrates undoubted advantage of our TS heuristic over the current static assignment. The average saving is 24.70% for all (35) problem instances. In addition, the savings are consistent over the five working days, and our heuristic uses very reasonable CPU times to solve the problem. The overall solid performance and significant savings give our heuristic a high promise as an effective practical tool for solving the AGAP problem.

In the second test set, we compare the performance of our heuristic with the exact method. We use CPLEX 6.0, a cutting-edge software for solving the MIP prob-

lem using branch and bound mechanism, to solve the 0-1 mixed formulation as specified by Model 2. Since this set of test problems have much smaller sizes, we slightly modify our TS heuristic to allow it to terminate early (e.g., at iteration 100). As pointed out previously, this set of test problems are generated more randomly. Unlike the first set of problems, there is no obvious feasible assignment which can be served as an initial solution. Therefore we start with an arbitrary assignment which may be infeasible. To address this, we calculate the infeasibility measure (e.g., $\sum_{i,j,k} max\{0, d_i + \beta - a_j\}$ $\forall i, j, k$ such that $z_{ijk} = 1$) of each move, multiplied by 10, and add it as a penalty to the move evaluation. Driven by this penalty, the search quickly yields feasible solutions. The comparisons are presented in Table 2 where the objective values and CPU times of both methods are reported.

Problem No.	Size ($n \times m$)	CPLEX		TS Solution	
		Cost	CPU (sec.)	Cost	CPU (sec.)
1	12 × 3	11275	105.48	11275	0
2	12 × 3	2793	4.94	2793	0
3	12 × 3	8434	7.08	8434	0
4	12 × 3	6250	7.88	6250	0
5	15 × 3	19529	618.74	19529	0
6	15 × 3	11025	294.18	11025	0
7	15 × 3	9804	43.11	9804	0
8	15 × 3	10875	44.31	10875	0
9	20 × 5	18958	122761.37	18958	0.01
10	20 × 5	19228	132776.56	19228	0.01

Table 2: Results on Second Set of Test Problem

Table 2 clearly shows that the branch-and-bound based exact method is impractical to solve the AGAP problem. The solution times required by CPLEX increase exponentially even for the indeed trivial problems. In contrast, our TS heuristic yields optimal solutions for all problem instance within negligible times.

5 Conclusions

In this paper, we consider an Airport Gate Assignment Problem that dynamically assigns airport gates to scheduled flights based on passengers' O&D flow data. The objective is to minimize the overall connection times that passengers walk to catch the connection flights. We first formulate this problem as a mixed 0-1 quadratic integer programming problem, then we reformulate it as a mixed 0-1 integer problem with a linear objective function and constraints. We develop a tabu search based heuristic for the AGAP problem that employs a standard version of short term memory, dynamic tabu tenure and aspiration criterion. We further enhance the heuristic by incorporating some highly effective candidate list strategies and intensification and diversification strategies.

We conduct two preliminary tests. First, we show that our TS heuristic can effectively achieve significant savings on passengers' connection time compared with the static gate assignment in current airline operation. The average saving for seven test problems with up to 400 flights and 50 gates over five consecutive working days is 24.70%. Then we show that the our heuristic yields optimal solutions for all instances that could be solved by an MIP-based exact method, while requiring only a fraction of the solution time.

Future improvements of our TS approach are anticipated to result by including frequency-based long term memory functions, and by using more refined candidate list strategies and other intensification and diversification strategies. Our model should incorporate various practical constraints and refined objective functions. Finally, the current AGAP model and algorithm is devised for the planning problem. To be a successful implementation in real world, our model and algorithm must be extended to handle the real-time gate assignment by patching the disrupted assigment caused by operations such as flight delays and cancellations.

Reference

1. Battiti, R. and G. Tecchiolli (1994), The Reactive Tabu Search, *ORSA Journal on Computing*, **6**, 126.
2. Dantzig, G.B. (1963), *Linear Programming and Extensions*, Princeton University Press, Princeton.
3. Glover, F. (1994), Unnoticed Ways to Improve Nonlinear Integer Programming Formulations, *Working Paper*, School of Business, University of Colorado, Boulder, CO80309.
4. Glover, F. (1996), Tabu Search and Adaptive Memory Programming – Advances, Applications and Challenges, in: *Interfaces in Computer Science and Operations Research*, Barr, Helgason and Kennington, eds., Kluwer Academic Publishers, 1-75.
5. Glover, F. and M. Laguna(1997), *Tabu Search*, Kluwer Academic Publishers.
6. Glover, F. and R.E.D. Woolsey (1974), Further Reduction of Zero-One Polynomial Programming Problems to Zero-One Linear Programming Problems, *Operations Research*, **Vol. 22, No.1**.
7. Xu, J., S. Y. Chiu and F. Glover (1996a) Using Tabu Search to Solve the steiner Tree-Star Problem in Telecommunications Network Design, *Telecommunication Systems*, **6**, 117-125.
8. Xu, J., S. Y. Chiu and F. Glover (1996b), Probabilistic Tabu Search for Telecommunications Network Design, *Combinatorial Optimization: Theory and Practice*, **Vol. 1, No. 1**, 69-94.
9. Xu, J., S. Y. Chiu and F. Glover (1997), Tabu Search for Dynamic Routing Communications Network Design, *Telecommunication Systems*, **8**, 55-77.

10. Xu, J., S. Y. Chiu and F. Glover (1998). Fine-Tuning a Tabu Search Algorithm with Statistical Tests, *International Transactions in Operational Research* **Vol. 5, No. 3**,233-244.

11. Xu, J., S. Y. Chiu and F. Glover (1999),Optimizing a Ring-Based Private Line Telecommunication Network Using Tabu Search, *Management Science*, **Vol. 45, No. 3**, 330-345.

12. Xu, J. and J. P. Kelly (1996), A New Network Flow-Based Tabu Search Heuristic for the Vehicle Routing Problem, *Transportation Science*, **30, 4**, 379-393.

13. Zangwill, W.I. (1965), Media Selection by Decision Programming, *Journal of Advertising Research*, **Vol.5, No.3**.