MIT Sloan School of Management

Working Paper 4388-01
December 2001

A VERY LARGE-SCALE NEIGHBORHOOD SEARCH ALGORITHM FOR THE COMBINED THROUGH AND FLEET ASSIGNMENT MODEL

Ravindra K. Ahuja, Jon Goodstein, Amit Mukherjee, James B. Orlin and Dushyant Sharma

A Very Large-Scale Neighborhood Search Algorithm for the Combined Through and Fleet Assignment Model

Ravindra K. Ahuja[1], Jon Goodstein[2], Amit Mukherjee[3], James B. Orlin[4], and Dushyant Sharma[5]

Abstract

The fleet assignment model (FAM) for an airline assigns fleet types to the set of flight legs that satisfies a variety of constraints and minimizes the cost of the assignment. A *through* connection at a station is a connection between an arrival flight and a departure flight at the station, both of which have the same fleet type assigned to them that ensures that the same plane flies both legs. Typically, passengers are willing to pay a premium for through connections. The through assignment model (TAM) identifies a set of profitable throughs between arrival and departure flights flown by the same fleet type at each station to maximize the through benefits. The through assignment model is usually solved after obtaining the solution from a fleet assignment model. In this current sequential approach, the through assignment model cannot change the fleeting in order to get a better through assignment, and the fleet assignment model does not take into account the through benefits. The goal of the combined through and fleet assignment model (ctFAM) is to come up with a fleeting and through assignment that achieves the maximum combined benefit of the integrated model. We give a mixed integer programming formulation of ctFAM that is too large to be solved to optimality or near-optimality within allowable time for the data obtained by a major US airline. We thus focus on neighborhood search algorithms for solving ctFAM, in which we start with the solution obtained by the previous sequential approach (that is, solving FAM first and followed by TAM) and improve it successively. Our approach is based on generalizing the swap-based neighborhood search approach of Talluri [1996] for FAM which proceeds by swapping the fleet assignment of two flight paths flown by two different plane types that originate and terminate at the same stations and the same times. An important feature of our approach is that the size of the neighborhood defined by us is very large; hence the suggested algorithm falls in the category of *Very Large-Scale Neighborhood (VLSN) Search Algorithms*. Another important feature of our approach is that we use integer programming to identify improved neighbors. We provide computational results which indicate that the neighborhood search approach for ctFAM provides substantial savings over the sequential approach of solving FAM and TAM.

[1] Department of Industrial and Systems Engineering, University of Florida, Gainesville, FL 32611, USA.
2 Information Services Division, United Airlines World Headquarters - WHQKB, Chicago, IL 60666, USA.
3 Information Services Division, United Airlines World Headquarters - WHQKB, Chicago, IL 60666, USA.
[4] Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA 02139, USA.
[5] Operations Research Center, Massachusetts Institute of Technology, Cambridge, MA 02139, USA.

1. INTRODUCTION

The airline industry has been a pioneer in using IE/OR techniques to solve complex business problems related to the schedule planning of the airline. Given a flight schedule, an airline's schedule planning group needs to decide the itinerary of each aircraft and each crewmember so that the total revenue minus the total operating costs is maximum and all the operational constraints are satisfied. The quality of the schedule is also measured in terms of other attributes such as schedule reliability during operations. The entire planning problem is too large to be solved to optimality as a single optimization problem using present day technology. Hence, it is typically divided into four stages (see, for example, Barnhart and Talluri [1997] and Gopalan and Talluri [1998]): (i) fleet assignment; (ii) through assignment; (iii) maintenance routing; and (iv) crew scheduling. These problems are solved sequentially where the optimal solution of one problem becomes the input for the following problem. There has been significant effort spent in modeling and solving these individual problems using advanced optimization models. The economies of scale at a large airline like United Airlines are such that a relatively minor improvement in contribution results in considerable improvement in the bottom line. As a result, airlines have benefited immensely from the advances in modeling these problems.

The next frontier in the optimization of schedule planning is in solving an integrated optimization problem that will consider the entire planning problem mentioned above and include other downstream issues that affect the overall schedule quality. Basically, the planning problem is a multi-criteria optimization problem, i.e., there are many objectives that have different metrics, different priorities and different constraints. A sequential approach to solve such problems has a major drawback in that the solution at each stage does not take into account the considerations of subsequent stages. This results in overall suboptimal solutions. For example, if a fleet assignment is performed without considerations of optimizing crew scheduling, it becomes an arbitrary input for the crew scheduling process. On the other hand, if the fleet assignment incorporates crew issues, then it is likely to provide a better starting point to the crew scheduling optimization, resulting in overall economic benefits for the airline.

The airlines are making a lot of effort to develop models to solve such integrated optimization problems for schedule planning. The objective is to break down the functional silos that exist in order to manage the complexity of the planning process by using advances in optimization technology and computing power. At United Airlines, there are two distinct strategies being pursued. The first strategy is to develop explicit joint optimization models with combined objective functions, combined set of constraints and combined data. Typically, these joint models are too large to be solved to optimality or near-optimality, suggesting that heuristics may be needed. Moreover, some downstream criteria cannot be represented easily in a form consistent with explicitly modeling the problem. For example, airline reliability is an important criteria in schedule planning that is related to the schedule structure, but it is very hard to model it in terms of a typical optimization problem. The second strategy exploits the nature of the planning problem. Instead of one optimal solution, there are typically many solutions that are close to the optimal in terms of contribution. However, these solutions can have very distinct characteristics on other criteria such as crew required, potential for through flights, schedule reliability, ground manpower requirements, etc. This implies that intelligent search techniques, when coupled with advanced

optimization modeling, hold a lot of promise in solving multi-criteria schedule planning problems. As a result, United Airlines initiated collaboration with MIT and University of Florida to explore solution techniques that exploited the nature of the overall schedule planning problem. The strategy was to build upon the foundation that already existed for modeling the individual stages and develop a robust and generic methodology that could be easily scaled up for other downstream criteria.

In this paper, we propose an integrated approach that first solves the separate models to optimality in a stage-wise fashion followed by solving the integrated model heuristically using neighborhood search techniques. This approach guarantees that the solution obtained by our approach is no worse than the solution obtained by the current sequential approach and in practice is better.

In this paper, we focus on integrating two of the airline scheduling models, the *Fleet Assignment Model* (FAM) and the *Through Assignment Model* (TAM), into a single model that we call the *Combined Through Fleet Assignment Model* (ctFAM). We next briefly describe these three models.

Fleet Assignment Model (FAM): In FAM, planes of different fleet type are assigned to flight legs to minimize the assignment cost and subject to the following three types of constraints: (i) *covering constraints*: each flight leg must be assigned exactly one plane; (ii) *flow balance constraints*: for each fleet type, the number of planes landing at a city must be equal to the number of planes taking off from the city; and (iii) *fleet size constraint*: for each fleet type, the number of planes used must not exceed the number of planes available. Abara [1989] and Hane et al. [1995] give an MIP formulation for FAM. Subsequently, Clarke et al. [1996] and Subramanium et al. [1994] provide extensions to incorporate additional operational constraints related to maintenance and crew scheduling.

Through Assignment Model (TAM): A *through connection* is a connection between an inbound flight leg and an outbound flight leg at a station which ensures that the same plane flies both legs. Both flights legs in a through connection get the same flight number in the airline's flight schedule. Since a through connection allows passengers to remain onboard instead of changing gates at busy airports, passengers are willing to pay a premium for such connections; this premium is termed as the *through benefit*. TAM takes as an input a list of candidate pairs of flight legs that can make through connections with corresponding through benefits, and identifies a set of most profitable through connections. Observe that we can make through connections only between flights flown by the same fleet type; hence the fleet assignment limits the possible through connections. In the current implementations, TAM takes as an input the fleet assignment, identifies inbound and outbound flights at each city flown by the same fleet type, and determines through connections (that must be a subset of the candidate pairs) to maximize the through benefit. This problem can be solved as a bipartite matching problem. However, in practice the solution must satisfy some additional constraints, which yields a constrained bipartite assignment problem that can be solved using MIP techniques. We refer the reader to the papers by Bard and Hopperstad [1987], Barnhart et al. [1998], Gopalan and Talluri [1998], and Jarrah and Reeb [1997] for additional details on the TAM.

The Combined Through Fleet Assignment Model (ctFAM): The through assignment depends on the fleet assignment in that a through connection requires that both its flight legs have the same fleet type. In

the current systems, FAM does not take into account the through benefits, and may yield fleet assignments with limited through assignment possibilities. TAM cannot change the fleeting in order to get a better through assignment. In our model, ctFAM solves the integrated model and simultaneously determines fleet assignments and through connections. The integrated model offers opportunities to obtain better solutions compared to the current sequential approach. We first developed an integer programming formulation of ctFAM, which was too large to be solved to optimality or near-optimality for a major US airline. We then pursued the approach outlined in Figure 1. In our approach, we first solve FAM to obtain an optimal (or nearly optimal) fleet assignment. For this fleeting, we then solve TAM to determine optimal (or nearly optimal) through connections. We then solve ctFAM heuristically using the neighborhood search algorithm with the optimal FAM and TAM solutions as the starting solution for the neighborhood search.

Neighborhood search algorithms are widely regarded as an important tool to solve difficult combinatorial optimization problems effectively. In recent years, the fields of operations research, mathematical programming, and computer science have all witnessed a strong interest in the development and analysis of neighborhood search based approaches. The primary reasons for the widespread application of neighborhood search techniques in practice are their intuitive appeal, flexibility and ease of implementation, and their excellent empirical results (see, for example, Aarts and Lenstra [1997], and Glover and Laguna [1997]). We decided to pursue neighborhood search algorithms for ctFAM for the following reasons:

(i) Neighborhood search algorithms have been very successful in solving a variety of large-scale combinatorial optimization problems.

(ii) Neighborhood search algorithms permit us to start with the excellent solution obtained by solving FAM first followed by solving TAM. This guarantees that the neighborhood search algorithm obtains a solution that is at least as good as that obtained by FAM followed by TAM, and possibly better.

(iii) Neighborhood search algorithms are typically quite efficient and scalable. Often, we can solve problems with only a linear increase (or a small polynomial increase) in the computation time.

(iv) Neighborhood search algorithms are often flexible enough to incorporate other constraints that are difficult to model through linear constraints.

We now present a brief overview of our neighborhood search algorithm for ctFAM. An issue of critical importance in a neighborhood search algorithm is the manner in which we define the neighborhood of a solution. As per Talluri [1996], we define neighbors of a given solution by performing "A-B swaps" for two specified fleet types A and B. An A-B swap consists of changing fleet types of some legs from A to B and of some legs from B to A so that all constraints remain satisfied. A profitable A-B swap decreases the total cost of the solution, which in our case includes the costs of throughs. Identifying a profitable A-B swap is not a trivial problem because the number of possible A-B swaps is exponentially large. Hence the neighborhood search algorithm using A-B swaps falls within the category of *Very Large-Scale Neighborhood (VLSN) Search Algorithms,* a topic studied by Thompson and Orlin

[1989], Thompson and Psaraftis [1993], and Ahuja et al. [2001a, 2001b]. The paper by Ahuja et al. [2002] presents a survey of very large-scale neighborhood search algorithms.

Our approach consists of determining the starting solution by first solving FAM followed by TAM. This solution is successively improved by our neighborhood search algorithm. In each iteration, the neighborhood search algorithm selects any two fleet types, which we label as A and B, and performs a profitable "A-B swap". An *A-B swap* consists of changing some legs flown by fleet type A to fleet type B, changing some legs flown by fleet type A to fleet type A, and changing some through connections appropriately. The number of A-B swaps can be very large and difficult to enumerate explicitly. We describe a method using *A-B Improvement Graphs* which allows us to obtain profitable A-B swaps quickly in practice. The A-B improvement graph is constructed in a manner that each negative cost directed cycle in the graph satisfying some constraints defines a profitable A-B swap.

The neighborhood search algorithms constructs the A-B improvement graph and solves an integer programming problem to identify a negative cost constrained directed cycle. This cycle yields a new fleet and through assignment with a lower cost. We repeat this process for every pair of fleet types A and B, and terminate when for every such pair of fleet types, we do not find improved neighbors. We developed and implemented both a local improvement algorithm (where we always perform cost-decreasing iterations) and a tabu search algorithm (where we sometimes allow cost-increasing iterations too). Our local improvement algorithm obtains a local optimal solution for ctFAM in 5-6 seconds, whereas we ran the tabu search algorithm for 20-25 minutes on our data sets, which are of realistic size data. The solutions obtained by our algorithms resulted in savings of $5 million to $25 million on an annual basis on the data provided by United Airlines. These results suggest that neighborhood search is a useful supplement to the techniques already developed in airline operations research.

One of our major contributions in this paper is to generalize Talluri's [1996] concept of A-B improvement graph so that it incorporates through constraints. Our approach, when specialized to FAM, provides a neighborhood that contains the neighborhood due to Talluri and is much larger. Moreover, Talluri's neighborhood made the following assumption concerning arrival and departure "banks": any plane arriving at the bank would have sufficient time to be assigned to any of the departures at the bank. This assumption is overly restrictive in practice. Our neighborhood structure does not make this assumption. Indeed, it does not even depend upon the existence of arrival and departure banks.

This paper is organized as follows. In Section 2, we present an integer programming formulation for ctFAM. Sections 3 develops our neighborhood search algorithms for ctFAM. We describe some implementation details in Section 4. We give our computational results in Section 5. Section 6 gives conclusions of our research and avenues of future research.

2. AN INTEGER PROGRAMMING FORMULATION OF CTFAM

In this section, we present an integer linear (IP) formulation of ctFAM. We formulate this problem as a flow problem on a network, which we call the *connection network*. We first present input data for ctFAM, followed by the description of the connection network, followed by the IP formulation.

Input Data:

We have the following input data for ctFAM:

L : The set of all flight legs which need to be assigned planes. We use the index i to represent a particular leg.

F : The set of all fleet types. We use the index f to represent a particular fleet type.

T : The set of all candidate through connections. Each through connection is specified by a pair (i, j) of flights.

size(f) : The number of planes of fleet type f available for assignment.

dep-time(i) : The departure time for flight leg i.

arr-time(i) : The arrival time for flight leg i. We denote by *arr-time*(i) as the time when flight i actually arrives plus the turn time (the time need to prepare the plane to be assigned to the next flight[*]). Thus, the plane released from the flight i can be assigned to any flight j with *dep-time(j)* ≥ *arr-time(i)* at the same city.

dep-city(i) : The departure city for flight leg i.

arr-city(i) : The arrival city for flight leg i.

c_i^f : The cost incurred in assigning fleet type f to flight leg i.

d_{ij}^f : The cost incurred in connecting flight leg i with the flight leg j provided *arr-city*(i) = *dep-city*(j) and both the legs are flown by the fleet type f. Observe that $d_{ij}^f < 0$ for $(i, j) \in T$, and 0 otherwise.

count-time : A time instant on the 24-hour time scale when no plane leaves or arrives, that is, *count-time* ≠ *arr-time*(i) or *dep-time*(i) for any $i \in L$. We will assume here that count-time is midnight.

Connection Network:

We now explain how to construct the connection network, which will be the basis of our integer programming formulation as well as our neighborhood search algorithm for ctFAM. We denote the connection network as $G = (N, E)$ where N denotes the node set and E denotes the arc set. The node set $N = \{i : i \in L\}$ is obtained by defining a node for each flight leg $i \in L$, and the arc set $E = \{(i, j)$: *arr-city*(i) = *dep-city*(j)$\}$ consists of all possible connections between inbound and outbound flight legs. Obviously, a

[*] In practice, the turn time also depends upon the fleet type but for the simplicity of notation, we assume it to be independent of the fleet type.

connection between flight legs i and j is possible only if the arrival city of leg i is the same as the departure city of leg j. We give an example of connection network in Figure 2.

A connection arc (i, j) is said to be a *through connection arc* if $(i, j) \in T$ and a *regular connection arc* otherwise. We will use the following additional notation related to the connection network:

$I(i) = \{(j, i) \in E: j \in N\}$,

$O(i) = \{(i, j) \in E: j \in N\}$,

$S = \{(i, j) \in E:$ *arr-time*$(i) <$ *count-time* $<$ *arr-time*$(j)\} \cup \{(i, j) \in E:$ *dep-time*$(i) <$ *count-time* $<$ *arr-time*$(i)\}$, where the inequalities are based on the circular 24-hour time.

The set $I(i)$ denotes the set of incoming arcs at node i in the connection network, the set $O(i)$ denotes the set of outgoing arcs, and S denotes the set of arcs in the connection network that cross the *count-time*, which we assume to be midnight. We call the arcs in S as *overnighting* arcs; it contains the set of connection arcs that cross the count-time and also those connection arcs whose arrival flights are in the air at count-time.

Decision Variables:

We define two sets of decision variables in our integer programming formulation. The first set of decision variables (y_i^f) specify the fleet assignment and the second set of decision variables (x_{ij}^f) specify the (regular or through) connection assignment.

y_i^f : This variable takes value 1 if the flight leg i is assigned fleet type f, and 0 otherwise.

x_{ij}^f : This variable takes value 1 if both the flight legs i and j are flown by the fleet type f and we make a (regular or through) connection between the flight legs i and j, and 0 otherwise.

Integer Programming Formulation:

We give below the integer programming formulation of ctFAM.

$$\text{Minimize} \sum_{i \in N} \sum_{f \in F} c_i^f y_i^f + \sum_{(i,j) \in E} \sum_{f \in F} d_{ij}^f x_{ij}^f \qquad (1a)$$

subject to

$$\sum_{f \in F} y_i^f = 1, \qquad \text{for all } i \in N \qquad (1b)$$

$$\sum_{(i,j) \in O(i)} x_{ij}^f = y_i^f, \qquad \text{for all } i \in N \text{ and all } f \in F \qquad (1c)$$

$$\sum_{(i,j)\in I(j)} x_{ij}^f = y_j^f, \qquad \text{for all } j \in N \text{ and all } f \in F \tag{1d}$$

$$\sum_{(i,j)\in S} x_{ij}^f \le size(f), \qquad \text{for all } f \in F \tag{1e}$$

$$x_{ij}^f \in \{0, 1\}, \qquad \text{for all } (i, j) \in E \text{ and for all } f \in F \tag{1f}$$

$$y_i^f \in \{0, 1\}, \qquad \text{for all } i \in N \text{ and for all } f \in F \tag{1g}$$

We represent a feasible solution of ctFAM as (x, y). The first and second terms in the objective function (1a) represent the contributions resulting from the fleet assignment and through assignment, respectively. The constraint (1b) ensures that each flight leg is assigned exactly one fleet type. The constraints (1c) and (1d) together with (1b) imply that each flight leg is assigned to another flight leg using a connection arc, and the two flight legs and the connection arc are assigned the same fleet type. The constraint (1e) ensures that the total number of planes of fleet type f in the assignment, which is the sum of the flows on arcs in S, is no more than the available planes given by $size(f)$. Observe that to compute the number of planes of a particular fleet type k used in a fleet schedule, we sum the flow of planes of that fleet type on the *overnighting* arcs.

In practice, the solution of ctFAM must also satisfy several additional constraints. These constraints incorporate aspects of maintenance routing and crew scheduling. To simplify the exposition, we defer the detailed description of these constraints to Appendix I. In Section 4, we explain how our algorithm needs to be modified to account for these constraints.

Though ctFAM can be formulated as an integer programming problem, this problem is too large to be solved to optimality or near-optimality (using the current LP technology) for the national network of a large US airline. In the data supplied by United Airlines, there were 1,609 flight legs and 13 fleet types. The resulting IP formulation had approximately 100,000 integer variables and 18,000 constraints. We could not solve problems of this magnitude using the commercial IP solvers. We then focused on neighborhood search algorithms to solve ctFAM. Additional reasons for considering neighborhood search algorithms have earlier been described in Section 1. We describe our neighborhood search algorithm in the next section.

3. NEIGHBORHOOD SEARCH ALGORITHMS FOR CTFAM

For any feasible solution (x, y) of ctFAM, we will define a set $\mathcal{N}(x, y)$ of neighboring solutions. Our neighborhood search algorithm works as follows:

algorithm *neighborhood search*;
begin
obtain an initial feasible solution (x, y) of ctFAM;
while there is a neighbor $(x', y') \in \mathcal{N}(x, y)$ with $c(x', y') < c(x, y)$ **do**
begin
replace (x, y) by (x', y');
end;
output (x, y), which is a locally optimal solution;
end;

Figure 2. A neighborhood search algorithm for ctFAM.

Though the neighborhood search algorithm stated above always replaces (x, y) by an improved neighbor (x', y'), there exist variants which do occasionally replace (x, y) by worse neighbors too. These variants include simulated annealing and tabu search algorithms. (We note that simulated annealing is impractical when the neighborhoods are exponentially large.) We have also investigated tabu search algorithms for ctFAM.

There are three primary steps involved in designing a neighborhood search algorithm for ctFAM: (i) creating an initial feasible solution (x, y); (ii) defining the neighborhood $\mathcal{N}(x, y)$ with respect to the solution (x, y); and (iii) searching the neighborhood $\mathcal{N}(x, y)$ to identify an improved solution. We shall now discuss these steps in greater detail.

Creating an Initial Feasible Solution:

A feasible solution for ctFAM consists of a feasible fleet assignment and a feasible set of connections. We first obtain a feasible fleet assignment by solving the fleet assignment model (FAM). The solution of the FAM gives us a fleet assignment assigning a plane type to each flight leg. We next use the through assignment model (TAM) to generate a set of connections. We solve the through assignment problem at each city for each fleet type and optimally match the inbound flight legs with the outbound flight legs flown by the same fleet type. We solve these bipartite matching problems to generate a set of connections. The solution thus obtained is the starting solution for our neighborhood search algorithm.

A-B Solution Graph:

The *A-B solution graph*, $S^{AB}(x, y)$, is a subgraph of the connection network $G = (N, E)$ and is defined with respect to a given fleeting and connection solution (x, y) and a pair of fleet types A and B. Its node set, $N(S^{AB}(x, y))$, and arc set, $E(S^{AB}(x, y))$, are defined as follows:

$$N(S^{AB}(x, y)) = \{i \in N:\ y_i^A = 1 \text{ or } y_i^B = 1\},$$
$$E(S^{AB}(x, y)) = \{(i, j) \in E:\ x_{ij}^A = 1 \text{ or } x_{ij}^B = 1\}.$$

In other words, the A-B solution graph $S^{AB}(x, y)$ is the subgraph of G whose node set comprises of the flight legs that are assigned fleet types A and B in the solution (x, y), and the arc set comprises of the

connections between those flight legs. We shall refer to a node in the *A-B* solution graph as an *A-node* if $y_i^A = 1$ and *B-node* if $y_i^B = 1$. We shall refer to an arc in the *A-B* solution graph as an *A-arc* if $x_{ij}^A = 1$ and *B-arc* if $x_{ij}^B = 1$.

A-B Swaps:

Our neighborhood search structure uses the concept of *A-B swaps* to define neighboring solutions. We first define an *A-B swap* in a very general manner, one that permits a much larger neighborhood than we subsequently search. Given a feasible solution (x, y) of ctFAM, and a pair of fleet types A and B, we say that (x', y') is an *A-B neighbor* of (x, y) if it is a feasible solution that differs only in the assignment of A-flights and B-flights. The operation of obtaining an A-B neighbor is called an *A-B swap*. Figure 3(a) shows a part of the solution graph $S^{AB}(x, y)$ and Figure 3(b) shows the same part after the A-B swap has been performed. In the figure, we show A-nodes and A-arcs using regular lines, and B-nodes and B-arcs using dashed lines. Observe that the A-B swap changes the fleet type of nodes 4 and 10 from A to B and changes the fleet type of nodes 3 and 6 from B to A. Changing the fleet types of these nodes requires changing the connections too because we can connect nodes with the same fleet type only. The A-B swap must also ensure that the connections can be feasibly made, that is, for each connection arc (i, j), the arrival time of flight i is less than the departure time of flight j.

Recall that while defining A-B swaps we require that we do not violate fleet size constraints for fleet types A and B. Figure 4(a) shows a part of the A-B swap where the number of planes of a particular type used can increase. Suppose that flights 1 and 3 are flown by fleet type A and flights 2 and 4 are flown by fleet type B. Assume that flights 1 and 2 arrive at times 2 PM and 4 PM, respectively, and the flights 3 and 4 depart at 5 PM and 3 PM, respectively. Since flight 1 connects to flight 3 which leaves three hours later, the arc (1, 3) is not an overnighting arc. However, flight 2 arrives at 4 PM and connects to flight 4, which departs at 3 PM. Thus the arc (2, 4) is an overnighting arc. If we change the fleet types of flights 1 and 3 from A to B, and of flights 2 and 4 from B to A, as shown in Figure 4(b), then we increase the number of planes used for type A by one and decrease the number of planes used for type B by one. Our neighborhood search algorithm does not allow such swaps if it leads to infeasibilities. Note that if we change the fleet type of flight 1 from A to B, fleet type of flight 2 from B to A, and swap their connections, as shown in Figure 4(c), then both the new connection arcs (1, 4) and (2, 3) are not overnight arcs. This swap will reduce the number of plane used for type B by one. Our neighborhood search algorithm allows such swaps.

The example shown in Figure 4 illustrates a very simple A-B swap; on the other hand, there can exist far more complex *A-B* swaps, that affect many more flights and connections. In principle, we could identify an improving *A-B neighbor* of (x, y) by solving a restricted integer program. We decided that this was computationally too intensive, and adopted a more efficient approach. We search a subset of *A-B neighbors* of (x, y) using network optimization. We next define the concept of *A-B* improvement graph, which allows us to efficiently identify profitable *A-B* swaps over a structured subset of the A-B neighborhood.

***A-B* Improvement Graph:**

Before we discuss the creation of our improvement graph, we note that the *A-B* solution graph satisfies the following cycle-based property: The solution graph as restricted to the A nodes is a union of node-disjoint cycles, and the solution graph is also the union of node-disjoint cycles. Equivalently, each A node i has exactly one outgoing arc and exactly one incoming arc, and both these arcs have A-nodes as the other endpoint. In our swaps, we will be changing some *A*-nodes to *B* nodes and vice-versa. We will construct our *A-B* network in such a way that an improving cycle leads to a new solution with the above cycle-based property.

Let us first illustrate the simplest type of swap before moving to the more complex swaps permitted below. Consider two directed paths P and P' in the A-B solution graph both starting at the same time t and the same location L, and both ending at the same time t' and the same location L', and such that P consists of A-flights and P' consists of B-flights. We can swap P and P', making all the flights of P into B-flights and making all of the flights of P' into A flights. To identify such path pairs, we could look for all paths of A-flights and all paths of B-flights starting at time t at location L and ending at time t' at location L'. Talluri [1996] recognized that we could find these paths in a simpler manner by reversing the direction of all B arcs, and then looking for a directed cycle. By doing so, one also identifies many other cycles, but each of the cycles (if midnight arcs are excluded from the cycles) corresponds to a valid A-B swap. In our approach, we also reverse the arcs incident to B nodes.

An *A-B* improvement graph, $G^{AB}(x, y)$, is constructed for a given fleeting and through solution (x, y) and a pair of fleet types *A* and *B*. Each arc (i, j) in the *A-B* improvement graph has an associated cost c_{ij}. The *A-B* improvement graph satisfied the property that each directed cycle in it satisfying some constraints, called the *validity constraints*, corresponds to an *A-B* swap with respect to the solution (x, y), and the cost of the directed cycle equals the change in the fleeting and through costs. Consequently, a negative cost directed cycle satisfying the validity constraints gives a profitable *A-B* swap. We will subsequently refer to a directed cycle in $G^{AB}(x, y)$ satisfying validity constraints as a *valid cycle*.

The node set of the *A-B* improvement graph is identical to that of the *A-B* solution graph. Hence it consists of *A*-nodes and *B*-nodes. Each arc (i, j) in the improvement graph signifies that we switch the fleet types of nodes i and j from *B* to *A* or from *A* to *B* (whichever is applicable) and reconnect the flights so that the connections are between flights that are assigned the same fleet types. In our approach, we add an arc (i, j) to the improvement graph whenever this change can be feasibly made without increasing the total plane count at the city *arr-city*(i) if i is A-node and *dep-city*(i) if i is a B-node. We define the cost c_{ij} of the arc (i, j) to be the change in the fleeting and through costs resulting from the change. Figure 5 summarizes the six types of arcs that can be added to the improvement graph. In the figure, we show an *A*-node or an *A*-arc using regular lines, and a *B*-node or *B*-arc using dashed lines. The detailed explanation of these arcs is given next.

Type 1 Arcs: Consider an arc (i, j) in the *A-B* solution graph which is an *A*-arc such that $(i, j) \notin$ S. We introduce the arc (i, j) in the improvement graph which corresponds to changing the plane types of both the flights i and j from *A* to *B*. Both flights i and j become *B* flights, and we assume that their connection

is maintained. The cost of the arc (i, j), c_{ij}, is the sum of (i) the change in the fleeting cost when plane type of flight i is changed from A to B, and (ii) the change in the through revenues of the connection (i, j) due to change in fleeting types. Notice that when computing c_{ij} we include the change in the fleeting cost of flight i only but not flight j. We do it because if we include the cost of changing the fleet types of both the nodes i and j in the cost of arc (i, j), then when we sum the cost of arcs in a valid cycle, we will be double counting the changes in the fleeting costs. Since the arc (i, j) does not belong to the set S, it does not affect the fleet size constraint.

Type 2 Arcs: A type 2 arc (j, i) is introduced in the improvement graph for each B-arc (i, j) in the A-B solution graph such that $(i, j) \notin$ S. This arc corresponds to changing the plane types of both the flights i and j from B to A and preserving the connection between the two flights. Notice that contrary to the case of type 1 arcs, we introduce the arc (j, i) instead of arc (i, j). The arcs are reversed as per the discussion above. The cost of the arc (j, i) captures the change in the fleeting cost of flight j and through costs of the connection arc (i, j).

Type 3 Arcs: A type 3 arc (i, l) is introduced in the improvement graph for every pair, (i, j) and (k, l), of A-arcs in the A-B solution graph such that the change corresponding to it does not violate the fleet size constraints. The arc (i, l) signifies changing the fleet types of both the flights i and l from A to B. Since we can make connections between flights flown by the same fleet type, this change requires changing the connections too; we thus need to reconnect flight i with flight l and flight k with flight j. The cost of the arc (i, l) captures the change in the fleeting cost of flight i and the change in the through costs due to reconnections. We point out that we add the arc (i, l) to the improvement graph only if the corresponding change does not increase the number of planes of type A and B. For example, we require that (i) $(i, l) \notin$ S, and (ii) if $(k, j) \in$ S than either $(i, j) \in$ S or $(k, l) \in$ S or both. Observe that in the absence of requirement (i), the number of planes of type B used could increase by 1 after the addition of arc (i, l). Similarly, if requirement (ii) is not satisfied than the number of planes of type A could increase by 1.

Type 4 Arcs: We introduce a type 4 arc (j, k) in the improvement graph for every pair of B-arcs (k, l) and (i, j) in the A-B solution graph such that flight k can connect to flight j and flight i can connect to flight l, and such that the change corresponding to it does not violate the fleet size constraints. The cost of the arc (j, k) includes the costs of changing fleet type of flight j and k from B to A and the change in the through costs due to the reconnections. Notice that a type 4 arc is similar to a type 3 arc except that the direction of the arc is reversed. The requirements on the connection arcs are the same as those in type 3 arcs.

Type 5 Arcs: We introduce a type 5 arc (i, k) in the improvement graph for every pair of arcs (i, j) and (k, l) in the A-B solution graph such that (i, j) is an A-arc, (k, l) is a B-arc, and the change corresponding to (i, k) does not violate the fleet size constraint, and such that flight i can connect to flight l and flight k can connect to flight j. The arc (i, k) corresponds to changing the fleet type of leg i from A to B and of leg k from B to A. Changes in the fleet types require changing the through assignments too; leg i connects to leg l, and leg k connects to leg j after the swap. The cost of the arc (i, k) captures the cost of the change in the fleet assignment of leg i and the change in through connection costs due to the reconnections. To ensure that the number of planes of type A and B do not increase, we use the following two requirements on the connection arcs involved: (i) if $(i, l) \in$ S then $(k, l) \in$ S, and (ii) if $(k, j) \in$ S then $(i, j) \in$ S.

Type 6 Arcs: A type 6 arc is similar to a type 5 arc but with its orientation reversed. We introduce the arc (j, l) in the improvement graph for every pair of arcs (i, j) and (k, l) in the A-B solution graph such that (i, j) is a B-arc, (k, l) is an A-arc, and the change corresponding to it does not violate the fleet size constraints. In addition, we require that flight i can connect to flight l and flight k can connect to flight j. The cost of the arc (j, l) includes change in the fleeting cost of flight j and the change in through costs due to reconnections. To ensure that the number of planes of type A and B do not increase, the requirements on the connection arcs are the same as those in type 5 arcs.

We will identify A-B swaps by defining valid cycles which we define next. In the A-B solution graph, each node i is connected to a unique node j through the arc (i, j) and is also connected from a unique node k through the arc (k, i). For each node i, we define its "mate" as follows: (i) if i is an A-node and (i, j) is an arc in the A-B solution graph, then $mate(i) = j$; and (ii) if i is a B-node and (k, i) is an arc in the A-B solution graph, then $mate(i) = k$.

Valid Cycles: *A directed cycle W in the A-B improvement graph is said to be a valid cycle if it satisfies the following property for every node* $i \in W$: $mate(i) \notin W$ *unless* $(i, mate(i)) \in W$.

The intuitive reason we do not allow valid cycles to contain both the nodes i and mate(i) in the valid cycles unless (i, $mate(i)$) $\in W$ is as follows. The purpose of constructing the improvement graph is that a directed cycle in it defines an A-B swap and that the cost of the cycle equals the cost of the A-B swap. A directed cycle, which is not a valid cycle, cannot ensure this property. Consider, for example, a directed cycle W in the improvement graph which contains a Type 5 arc (i, k) (see Figure 5). Let node j = $mate(i)$ and $l = mate(k)$. The arc (i, k) *signifies* the change that flight i reconnects to flight l and flight k reconnects to flight j, and the cost of the arc (i, k) captures the cost of these changes. If we allow the cycle W to visit node j or node l, then we will not be able to preserve the change indicated by arc (i, k) and its cost will become incorrect. Thus, if we make arc (i, k) part of the cycle, then we must disallow the mates of these nodes from being a part of the cycle. This difficulty arises when we include arcs of Type 3, 4, 5, or 6 in the cycle W. This difficulty does not arise when we make an arc of Type 1 or Type 2 to be the part of the cycle in which case we include both the node i and its mate. Hence the "unless" clause in the definition of the valid cycle.

We will now give a numerical example that a valid cycle in the improvement graph gives an A-B swap; this example will be followed by a formal proof of the general result. Consider the part of the A-B solution graph shown in Figure 6(a). When we construct the improvement graph, it will contain the valid cycle W = 3-4-10-7-8-6-3 shown in Figure 6(b). This cycle denotes the A-B swap, which when performed, produces the solution shown in Figure 6(c). Observe that all the nodes in the cycle switch their fleeting types. The arc (3, 4) in the valid cycle W is a Type 6 arc, this arc signifies that nodes 3 and 4 switch their fleeting types and the inbound flights into these nodes swap their connections. The next arc (4, 10) in the cycle is a Type 3 arc which changes fleeting types and connections. The next arc (10, 7) is a Type 1 arc; it only changes the fleeting. The next arc (7, 8) in the cycle is a Type 5 arc which captures the fact that the outbound flights from these two nodes swap their flights. Finally, the two arcs (8, 6) and (6, 3) are Type 2 arcs which change the fleeting types but not the connections. Figure 6(c) shows the same part of the solution graph when the corresponding A-B swap has been performed.

We are now ready to prove the general result.

Theorem 1. *Each valid cycle in the A-B improvement graph $G^{AB}(x, y)$ gives an A-B swap with respect to the solution (x, y).*

Proof: We note that any A-B swap results in a solution satisfying the constraints (1b) since any flight leg that has fleet types A or B assigned to them will have a fleet type (A or B) after the swap. The constraints (1e) are also satisfied since the changes corresponding to each arc in the A-B improvement graph ensure that the fleet size constraints (1e) are satisfied. We shall now show that the constraints (1c) and (1d) are also satisfied. This amounts to showing that the cycle-based property is maintained by the swap. Let W denote the valid cycle. Let i be a node of the A-B solution graph. We assume inductively that node i has one incoming arc and one outgoing arc in the current solution, and these arcs join node i to nodes of the same fleet type. We want to prove that this property is satisfied after the A-B swap. Our proof relies on the consideration of a number of cases. We show that the property holds for the A-nodes affected by the swap. A similar argument can be made for the B-nodes.

Suppose first that $i \in W$ and that i is an A-node. We consider first the node that directly follows node i in W. We will show that after the swap, there is a B-node that directly follows node i in the resulting A-B solution graph. If (i, j) is of type 1, then arc (i, j) is a B-arc in the A-B solution graph after the swap. If (i, *l*) is of type 3, then arc (i, *l*) is a B-arc in the solution graph after the swap. If arc (i, k) is of type 5, then (i, *l*) is a B-arc in the solution graph after the swap. We also note that cases 2, 4 and 6 are not applicable to the arcs leaving an A-node.

We now consider the node that directly precedes an A-node r in W. We will show that after the swap, there is a B-node that directly precedes node r in the resulting A-B solution graph. If (i, j) is of type 1 (in this case, r = j), then (i, r) is a B-arc in the A-B solution graph after the swap. If (i, *l*) is of type 3, (in this case, r = *l*), then (i, r) is a B-arc in the A-B solution graph after the swap. If (j, *l*) is of type 6, (in this case, r = *l*), then (i, r) is a B-arc in the A-B solution graph after the swap. We have just established that for an A-node in W, there is exactly one outgoing B-arc and exactly one incoming B-arc after the swap. A similar argument can be made for the B-nodes in the cycle W.

We now consider nodes that are not in W and are affected by the swap. In cases 1 and 2, there are no such nodes. In case 3, node j has its incoming arc changed from (i, j) to (k, j), and node k has its outgoing arc changed from (k, *l*) to (k, j), and the cycle property remains satisfied after the swap. (We know that $j \notin W$, and $k \notin W$ because W is valid). In case 4, node *l* has its incoming arc changed from (k, *l*) to (i, *l*), and node i has its outgoing arc changed from (i, j) to (i, *l*), and the cycle property remains satisfied after the swap. (We know that $i \notin W$, and $l \notin W$, because W is valid.) In case 5, the A-node j has its incoming arc changed from (i, j) to (k, j), and the B-node *l* has its incoming arc change from (k, *l*) to (i, *l*), and the cycle property remains satisfied after the swap. (We know that $j \notin W$, and $l \notin W$, because W is valid.) Finally, in case 6, the B-node i has its outgoing arc changed from (i, j) to (i, *l*), and the A-node k changes its outgoing arc from (k, *l*) to (k, j), and the cycle property remains satisfied after the swap. (We know that $i \notin W$, and $k \notin W$, because W is valid.) This completes the proof of the theorem.♦

3.5 Identifying *A-B* swaps

In the last section, we have shown that we can identify *A-B* swaps by identifying valid cycles in the *A-B* improvement graph. However, identifying valid cycles in a graph is an NP-complete problem (see Appendix II). Fortunately, this problem was typically solved in a fraction of second using CPLEX in our benchmark cases. We will next model the problem of finding a union of node-disjoint valid cycles as an integer programming problem.

We first introduce some notation related to the integer program. Let $N' = N(G^{AB}(x, y))$ denote the set of nodes and $E' = E(G^{AB}(x, y))$ denote the set of arcs in the A-B improvement graph. We associate a binary variable w_{ij} with each arc $(i, j) \in E'$. This variable takes value 1 if arc (i, j) is present in some valid cycle, and takes value 0 otherwise. We give the IP formulation next followed by its explanation.

$$\text{Minimize} \quad \sum_{(i,j)\in E'} c_{ij} w_{ij} \tag{2a}$$

subject to

$$\sum_{\{j:(j,i)\in E'\}} w_{ji} - \sum_{\{j:(i,j)\in E'\}} w_{ij} = 0, \qquad \text{for } all\ i \in N', \tag{2b}$$

$$\sum_{\{j:(i,j)\in E'\setminus\{(i,mate(i))\}\}} w_{ij} + \sum_{\{j:(mate(i),j)\in E'\}} w_{mate(i),j} \le 1, \quad \text{for } all\ i \in N', \tag{2c}$$

$$w_{ij} \in \{0,1\}, \text{ for} (i, j) \in E' . \tag{2d}$$

In the above formulation (2), the constraints (2b) and (2d) imply that the solution is a 0-1 circulation. This 0-1 circulation can be decomposed into unit flows along directed cycles. The constraints (2c) ensure that the flow passing through each node *i* plus the flow passing through the node *mate*(*i*) is at most 1, which implies that the resulting flow will not pass through both the nodes *i* and *mate*(*i*). An exception to this rule occurs when flow takes place over the arc (*i*, *mate*(*i*)) in which case both the nodes *i* and *mate*(*i*) can be visited. It is easy to see that a feasible solution of (2) gives a set of valid cycles. If the improvement graph does not contain any negative cost valid cycles, then $w = 0$ will be an optimal solution of (2). If the improvement graph contains a negative cost valid cycle, then an optimal solution w^* of (2) will give a collection of valid cycles with the minimum total cost. Using flow decomposition (see, for example, Ahuja, Magnanti, and Orlin [1993]), we can decompose w^* into a set of node-disjoint cycles. Each of these cycles has a negative cost or a cost of 0. The negative cost cycles include an associated profitable A-B swap.

3.6 Neighborhood Search Algorithms

We are now in a position to describe our neighborhood search algorithm for ctFAM. Figure 7 describes the generic version of our algorithm. Our neighborhood search algorithm for ctFAM performs passes over all fleet pairs *A* and *B* and performs profitable *A-B* swaps. The algorithm terminates when in one complete pass it finds that no profitable swap exists for any pair of fleet types *A* and *B*.

```
algorithm ctFAM neighborhood search;
begin
    solve FAM to determine the optimal fleet assignment y;
    solve TAM to determine the optimal connections x for the fleet assignment y;
    repeat
        for each pair of the fleet types A and B do
        begin
            construct the A-B solution graph S^AB(x, y);
            construct the A-B improvement graph G^AB(x, y);
            while the A-B improvement graph G^AB(x, y) contains negative cost valid cycles do
            begin
                determine a set W of negative cost valid cycles in the A-B improvement graph;
                perform A-B swaps corresponding to W;
                update the A-B solution graph S^AB(x, y);
            end;
        end;
    until for every pair of fleet types A and B, G^AB(x, y) contains no negative cost valid cycle;
end;
```

Figure 7. The neighborhood search algorithm for ctFAM.

4. IMPLEMENTATION DETAILS

We now describe some important details of the implementation of our neighborhood search algorithm.

Identifying Negative Cost Valid Cycles: To identify a negative cost valid cycles in the *A-B* improvement graph, we solve the IP problem (2) using the commercial solver CPLEX 6.5 and do not run it up to optimality as it takes too much time. The solver solves the IP problem using a branch and bound algorithm. We keep track of the number of integer solutions found by the branch and bound algorithm and stop it as soon as it finds an optimal solution or finds 10 integer solutions, whichever occurs earlier. We use the best integer solution found, decompose it into node-disjoint profitable valid cycles, and perform *A-B* swaps corresponding to each valid cycle. Our neighborhood search needs only one negative cost valid cycle to improve the current solution and it need not be the best valid cycle. Consequently, we may terminate the IP whenever it has found a negative cost valid cycle.

Updating Flight Connections: Our neighborhood search algorithm starts with a solution where the flight connections (given by the solution x) are optimal for the specified fleet assignment (given by the solution y). Each *A-B* swap performed by the algorithm changes the fleet assignment of some flight legs and may also change flight connections. After this change, the modified flight connections x' may not be optimal for the modified fleet assignment y'. Hence, a possibility to improve the solution value exists by changing connections without changing the fleet assignment. Our algorithm checks for these possibilities and makes switches when improvements are possible. It solves a TAM for the fleet types *A* and *B* at every city where *A-B* swap has changed the fleet assignment. This step takes only a small proportion of the overall computational time, and occasionally improves the solution value substantially.

Handling Additional Constraints: In Section 2, we noted that the solutions of ctFAM also need to satisfy three additional set of constraints (A1) – (A3) described in Appendix I. To incorporate these additional constraints, we first added these constraints to FAM and to TAM so that the initial solution constructed by using these models satisfies these constraints. Subsequently, we ensured that each *A-B* swap performed by the algorithm maintains these additional constraints. We considered the straightforward approach of directly incorporating the constraints into the integer program in (2). Unfortunately, this method makes the integer program (2) substantially much more difficult to solve, and not practical for a neighborhood search approach. However, we also made the following fortuitous discovery: approximately half of all improving valid cycles for our previous model satisfy constraints (A1) - (A3). Accordingly, we solved (2) using the IP solver with no additional constraints. We performed an A-B swap as determined by the solution of the integer program when it also satisfied constraints (A1) – (A3).

Tabu Search: The algorithm described in Figure 7 is a pure local search algorithm. We also implemented a tabu search algorithm. (See Glover and Laguna [1997] for details on tabu search). We implemented a version of the tabu search that incorporated the short-term memory aspect of tabu search; that is, we used tabu lists. To ensure that the tabu search approach would generate at least one valid cycle at each iteration, we added the constraint $\Sigma_{(i,j)\in E} w_{ij} \geq 1$ to (2). In order to cut down on some of the unproductive searching, we restricted our search to a small subset of "promising" *A-B* pairs of fleet types. In addition, when we solved (2) by the IP solver, we enumerated 100 integer solutions only and the best solution among them determined the set of *A-B* swap performed. Each flight involved in an *A-B* swap is made tabu for the next 5 iterations. For any pair of fleet types *A* and *B*, we apply the tabu search algorithm for 100 iterations and record the best solution found.

5. COMPUTATIONAL TESTING

In this section, we present computational results of our neighborhood search algorithms. We programmed our algorithms in the C programming language and on a Pentium 4 1.4 GHz processor computer with 512MB RAM and a Linux operating system. We tested our algorithms on the data provided by United Airlines.

We tested our local improvement and tabu search algorithms on four problems whose data was provided by United Airlines: (i) FAM without maintenance constraints; (ii) ctFAM without maintenance constraints; (iii) FAM with maintenance constraints; and (iv) ctFAM with maintenance constraints. The starting solutions for these problems were obtained by solving integer programming problems. The integer programming problems were run up to 30 minutes as is the practice at that airline. The best integer solution obtained became the starting point of our neighborhood search algorithms. The table shown in Figure 8 gives the changes in FAM and through contributions by the use of neighborhood search. Our objective in the neighborhood search algorithms was to maximize the total fleet assignment and through contribution. The improvements obtained are reported on an annual basis. We observe both the local and tabu search algorithms improve the integer programming solutions quickly in a fairly reasonable time. The following additional conclusions can be drawn from the table:

- The local improvement algorithm is efficient and terminates quickly. It also improves the FAM solution. The reason that it can improve the FAM solution is that the FAM solution was not solved to optimality, but terminated with a nearly optimal solution. The neighborhood search algorithms found the possible improvements quickly. This suggests that neighborhood search algorithms can be used as a supplement to the integer programming techniques.

- Our algorithms for ctFAM with maintenance constraints improved the integer programming solution by a substantially larger amount than ctFAM without the maintenance constraints. Maintenance constraints make the integer programming problem harder and the solution produced by the IP software leaves more room for possible improvement which our neighborhood search algorithm is able to obtain.

- Our local improvement algorithm terminates in a matter of a few seconds. Our tabu search algorithm takes substantially longer then the local improvement algorithm, and is able to obtain somewhat better solutions. Our tabu search implementation was a straightforward implementation. We believe that greater savings can be obtained by a better and more sophisticated implementation of the tabu search.

We performed some additional computational results to assess the behavior of our algorithms. Figure 9 shows the statistics we noted for our four models.

6. CONCLUSIONS

In this paper, we study the combined through and fleet assignment model (ctFAM) which integrates the fleet assignment model (FAM) and the through assignment model (TAM). We give an integer programming formulation of ctFAM which, unfortunately, is too large to be solved to optimality or near-optimality using the state-of-art commercial IP solvers. We propose a swap-based neighborhood search algorithm for ctFAM that proceeds by swapping the fleet types of flights flown by two fleet types. Our swap based neighborhood structure generalizes the previous neighborhoods suggested by Berge and Hopperstad [1993] and Talluri [1996]. We searched our (exponentially large) neighborhood heuristically using an integer programming solver. We implemented two versions of our basic algorithm – a local improvement algorithm and a tabu search algorithm.

Preliminary computational results of our algorithms are quite encouraging. On the data provided by United Airlines, both our local improvement algorithm and our tabu search algorithm obtained substantial improvements in savings and computational times. The airline is currently converting the prototype into a full-scale application for use in the scheduling department. There are plans to expand the approach for solving more complex mutli-criteria optimization problems by incorporating other downstream criteria in the schedule planning process. In conclusion, a comprehensive framework has been developed for the airlines to model and solve advanced planning problems.

ACKNOWLEDGEMENTS:

The authors gratefully acknowledge support through NSF Grants # DMI-9900087 (at the University of Florida) and DMI-9820998 (at MIT) and a grant from United Airlines. We also thank Raj Sivakumar and Ram Narsimhan at United Airlines for their help and support during the course of this project.

REFERENCES:

Aarts, E., and J.K. Lenstra. 1997. *Local search in Combinatorial Optimization*. John Wiley.

Abara, J. 1989. Applying integer linear programming to the fleet assignment problem. *Interfaces* **19,** 20-28.

Ahuja, R.K., J.B. Orlin, D. Sharma. 2001a. Multi-exchange neighborhood search algorithms for the capacitated minimum spanning tree problem. *Mathematical Programming* **91**, 71-97.

Ahuja, R.K., J.B. Orlin, D. Sharma. 2001b. A composite neighborhood search algorithm for the capacitated minimum spanning tree problem. Submitted to *Operations Research Letters*.

Ahuja, R.K., O. Ergun, J.B. Orlin, and A.P. Punnen. 2002. A survey of very large-scale neighborhood search techniques. To appear in *Discrete Applied Mathematics*.

Ahuja, R.K., T.L. Magnanti, and J.B. Orlin. 1993. *Network Flows: Theory, Algorithms and Applications*. Prentice Hall, New Jersey.

Bard, J.F., and C.A. Hopperstad. 1987. Improving through-flight schedules. *IIE Transactions* **19**, 242-251.

Barnhart, C., N.L. Boland, L.W. Clarke, E.L. Johnson, G.L. Nemhauser, and R. Shenoi. 1998. Flight string models for aircraft fleeting and routing. *Transportation Science* **32,** 208-219.

Barnhart, C., and K.T. Talluri. 1997. Airlines operations research. *Design and Operation of Civil and Environmental Engineering Systems*, Edited by A. McGarity and C. ReVellepp, 435-469.

Berge, M.E., and C.A. Hopperstad. 1993. Demand driven dispatch: A method for dynamic aircraft capacity assignment, models and algorithms. *Operations Research* **41,** 153-168.

Clarke, L. W., C.A. Hane, E.L. Johnson, and G.L. Nemhauser. 1996. Maintenance and crew considerations in fleet assignment. *Transportation Science* **30,** 249-260.

Glover, F., and M. Laguna. 1997. *Tabu Search*. Kluwer Academic Publishers, Norwell, MA.

Gopalan R., and K.T. Talluri. 1998. Mathematical models in airline schedule planning: A survey. *Annals of Operations Research* **76**, 155-185.

Hane, C. A., C. Barnhart, E.L. Johnson, R.E. Marsten, G.L. Nemhauser, and G. Sigmondi. 1995. The fleet assignment problem: Solving a large-scale integer program. *Mathematical Programming* **70,** 211-232.

Jarrah, A.I.Z., and J.C. Reeb. 1997. An optimization model for assigning through flights. Technical Document, United Airlines.

Subramanium, R., R.P. Scheff Jr., J.D. Quillinan, D.S. Wiper, and R.E. Marsten. 1994. Coldstart: Fleet assignment at Delta Air Lines. *Interfaces* **24,** 104-120.

Talluri, K.T. 1996. Swapping applications in a daily fleet assignment. *Transportation Science* **31,** 237-248.

Thompson, P.M., and J.B. Orlin. 1989. The theory of cyclic transfers. Working Paper No. OR 200-89, Operations Research Center, MIT, Cambridge, MA.

Thompson, P.M., and H.N. Psaraftis. 1993. Cyclic transfer algorithms for multivehicle routing and scheduling problems. *Operations Research* **41**, 935-946.

APPENDIX I

In this appendix, we briefly discuss three kinds of additional constraints that are enforced on a fleet assignment, and are closely related to constraints faced in other fleet scheduling problems as well. They are related to maintenance and crew scheduling. The rational behind these constraints is discussed in detail in Clarke et. al. [1996]. Our ctFAM includes these additional constraints and solutions obtained by our algorithms satisfy these constraints.

1. ***Service Maintenance Constraints***. For each fleet type *f*, the service maintenance constraints specify a set of maintenance stations at which a certain desired percentage of aircraft of fleet type *f* must be on the ground at midnight. These constraints can be easily incorporated into the ctFAM formulation. Let S^f denote the set of connection arcs such that a plane using one of these arcs is on the ground at midnight at one of the maintenance stations for fleet type *f*. Let p_f denote the desired percentage of aircraft of fleet *f* at its maintenance stations. Using this notation, we can write the service constraint for fleet type *f* in the integer programming formulation (1) as: $\sum_{(i,j)\in S^f} x_{ij}^f \geq \frac{p_f}{100} size(f)$.

2. ***Aircraft Balance Check Constraints***. These constraints model the longer balance check maintenance (10-12 hours) done on the aircraft. An aircraft balance check constraint specifies a station *s*, an interval of the day (a, b), duration of the check *D*, a list *L* of fleet types, and a number *K* such that *K* planes from the fleet type list *L* must receive balance check of duration *D* within the interval (*a*, *b*) at station *s*. In order to incorporate this constraint in the integer program (1), let *Q* denote the set of connection arcs such that a plane taking any of the connections in *Q* is on the ground at station *s* for a duration of at least *D* units between the interval (*a*, *b*). The aircraft balance check constraint can then be specified as: $\sum_{(i,j)\in Q} \sum_{f\in L} x_{ij}^f \geq K$. There may be several such constraints, one per type of balance check constraint.

3. ***Crew Block Hour Constraints***. The *block hour* of a flight is the time that elapses between the flight leaving the gate at the departure city and entering the gate at the arrival city. The crew block hour constraint requires that the total *block hours* of all the flight legs that are assigned to a given subset of fleet types should be bounded. Since each crew is typically trained for a subset of aircraft types, these constraints ensure that none of the crews is over or under-utilized. For each flight node *i*, let b_i denote the block time of the flight leg associated with it. Let *L* represent the set of fleet types involved in the crew block hour constraint and *m*, *M*, respectively, denote the lower and upper bounds on the total block hours allowed for all the planes belonging to the fleet types in *L*. The crew block hour constraint can be incorporated in the formulation (1) as $m \leq \sum_{f\in L} \sum_{i\in N^c} b_i y_i^f \leq M$. There may be several such constraints, one per type of balance check constraint.

APPENDIX II

In this appendix, we show that the problem of finding a negative cost valid cycle in a graph is NP-complete.

Negative Cost Valid Cycle Problem:

Input: *A graph G = (N, A), arc costs c: A $\rightarrow$ R, and a function mate: N $\rightarrow$ N such that for i $\in$ N, (i, mate(i)) $\in$ A and mate(i) $\neq$ mate(j) for j $\neq$ i.*

Question: Is there a negative cost valid cycle W in G (that is, is there a cycle W such that for every node i $\in$ W: mate(i) $\notin$ W unless (i, mate(i)) $\in$ W)?

We refer to an input instance of the problem as a *yes instance* if the answer to the question is yes. It is easy to see that the negative cost valid cycle problem is in NP since a negative cost valid cycle is a succinct certificate for the yes instances. In order to prove the NP-completeness of the negative cost valid cycle problem, we shall provide a polynomial time transformation from another problem called the *path through forbidden pairs*, which is known to be an NP-complete problem (see, for example, Garey and Johnson [1979]).

Path through Forbidden Pairs Problem:

Input: *A graph G′ = ({1, 2,..., 2n}, A′) with 2n+1 nodes for some n > 0.*

Question: *Is there a path P from node 1 to node 2n+1 in G′ satisfying the following property: for each k = 1, 2, ..., n-1, the nodes 2k, 2k+1 can not simultaneously belongs to the path P?*

Given an input graph G' for the path through forbidden pairs problem, we construct an input instance of the negative cost valid cycle problem as follows. The input graph for our instance is G = (N, a), where $N = \{0, 1, 2, \ldots, 2n\}$, $A = A' \cup S \cup \{(2n, 1)\} \cup \{(0, 2n)\}$) and $S = \{(2k, 2k+1): k = 1, \ldots, n-1\} \cup \{(2k+1, 2k): k = 0, \ldots, n-1\}$. The arcs in the set S and the arc $(0, 2n)$ are added so that we can define the *mate* function in the desired manner. For $k = 1, 2, \ldots, n-1$, we define $mate(2k) = 2k+1$ and $mate(2k+1) = 2k$. We set $mate(0) = 2n$, $mate(2n) = 1$, and $mate(1) = 0$. The reader can verify that our *mate* function satisfies the required conditions. We define the arc costs c as follows:

$$c_{ij} = \begin{cases} 0 & \text{if } (i,j) \in A' \\ 1 & \text{if } (i,j) \in S \\ -1 & \text{if } (i,j) = (2n,1) \end{cases}$$

It is easy to see that the instance above can be constructed in time polynomial in the size of G'.

Theorem 2. *The instance (G, c, mate) is a yes instance for the negative cost valid cycle problem if and only if the graph G′ is a yes instance for the path through forbidden pairs problem.*

Proof: If a path P exists in G' from node 1 to node $2n$ such that it satisfies the required condition for path through forbidden pairs problem, then by the definition of c, the cost of the path P is 0 in the instance (G, c, *mate*). Further, by the construction of the *mate* function, the cycle obtained by adding the arc $(2n, 1)$ to path P is a valid cycle. Hence, if G' is a yes instance of the path through forbidden pairs problem, then (G, c, *mate*) is a yes instance of the negative cost valid cycle problem.

Conversely, if W represents a negative cost valid cycle in the instance (G, c, *mate*) then it must contain the arc $(2n, 1)$ and none of the arcs $(i, j) \in S$ can be in the cycle W. By the construction of the *mate* function, this implies that for $k = 1,\ldots, n\text{-}1$, the nodes $2k$ and $2k+1$ cannot be in the cycle W simultaneously. Hence, the path obtained by removing the arc $(2n, 1)$ from the cycle W satisfies the condition for the path through forbidden pairs problem in the graph G'. This proves our result. ♦

Using the previous theorem and our construction of the instance (G, c, *mate*), we observe that there is a polynomial time transformation from any instance of the path through forbidden pairs problem to an instance of the negative cost valid cycle problem. Therefore, the negative cost valid cycle problem is an NP-complete problem.

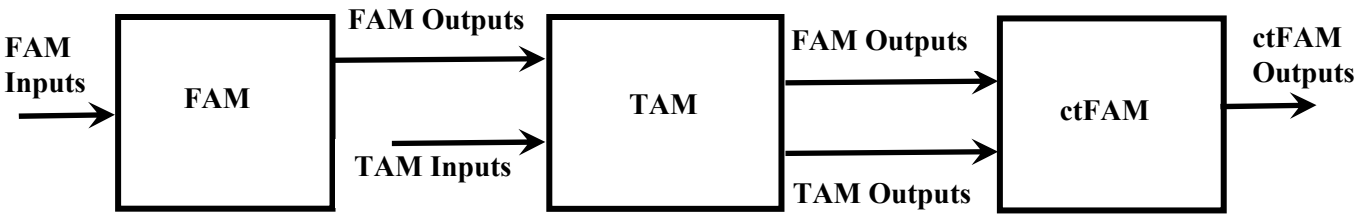


Figure 1. Our approach for solving ctFAM.

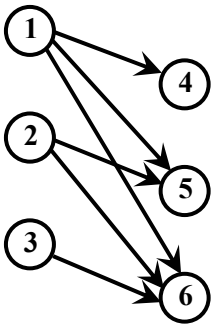


Figure 2. Part of the connection network at a city with the inbound flights 1, 2, and 3, and the outbound flights 4, 5, and 6.

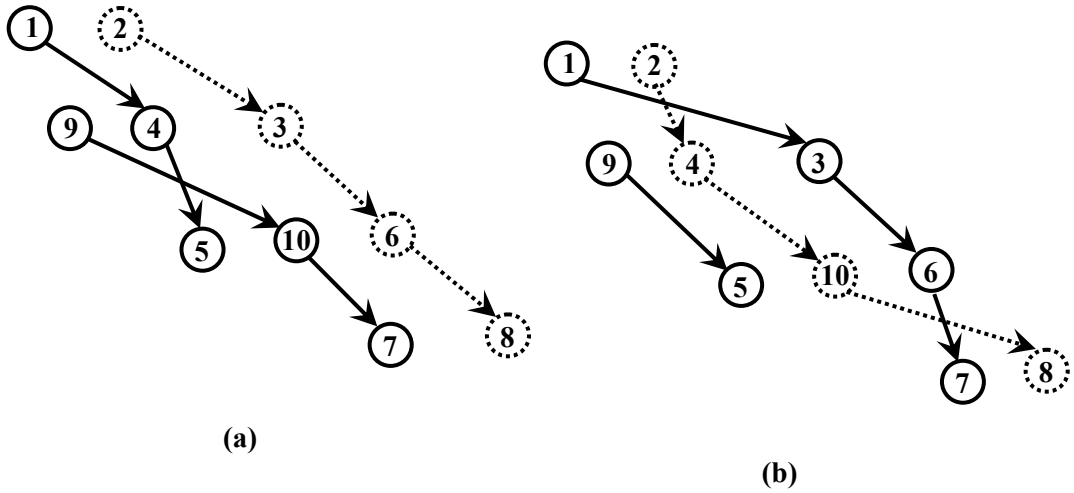


Figure 3. Illustrating an A-B swap.
(a) Part of the solution graph before the A-B swap.
(b) Part of the solution graph after the A-B swap.

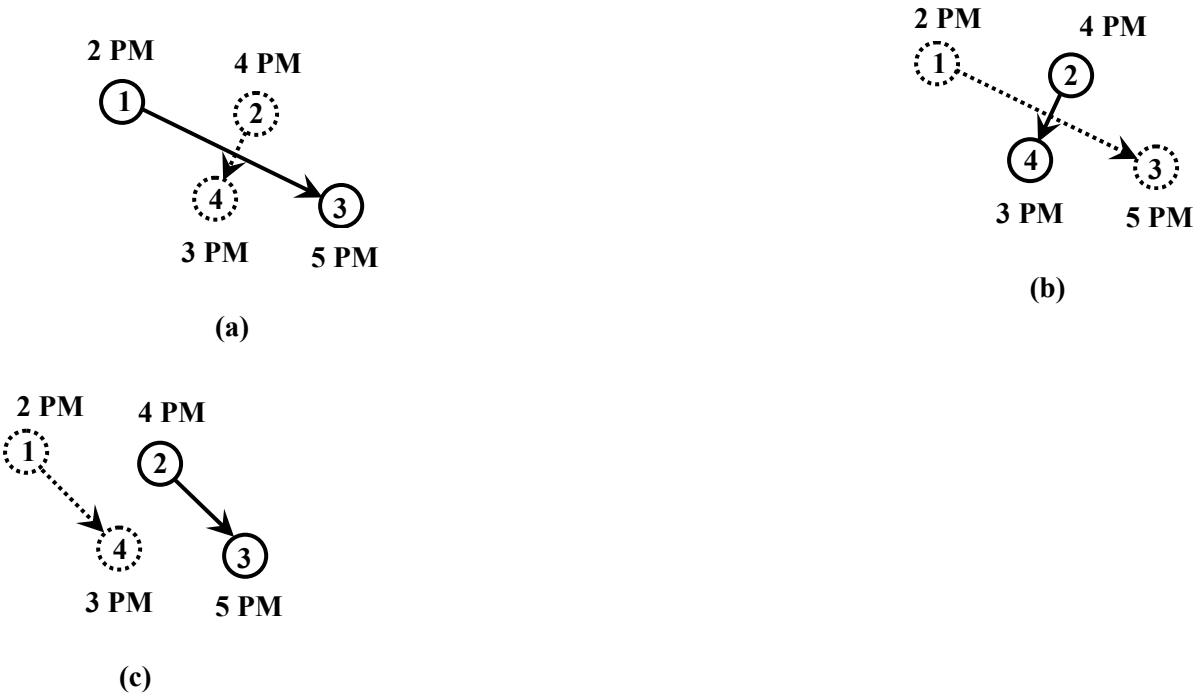


Figure 4. Effect of swaps on the number of planes used.
(a) Four flights and two connections. (b) a possible swap which increases the number of planes used for type A and decreases the number of planes used for type B; (c) a possible swap which decreases the number of planes used for type B.

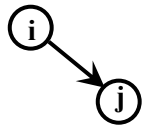
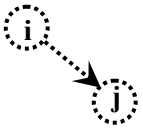
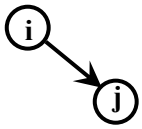
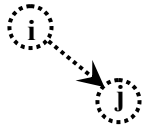
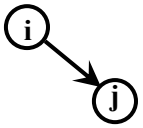
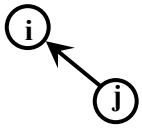
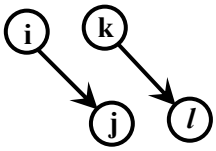
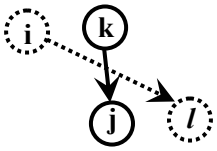
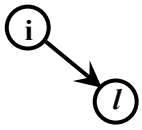
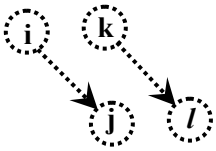
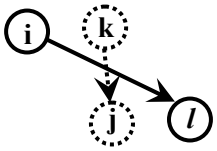
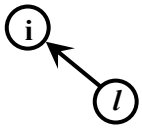
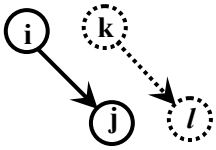
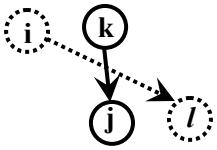
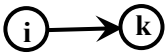
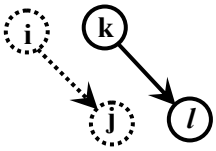
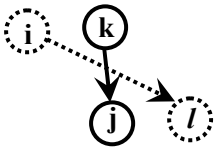
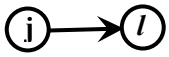
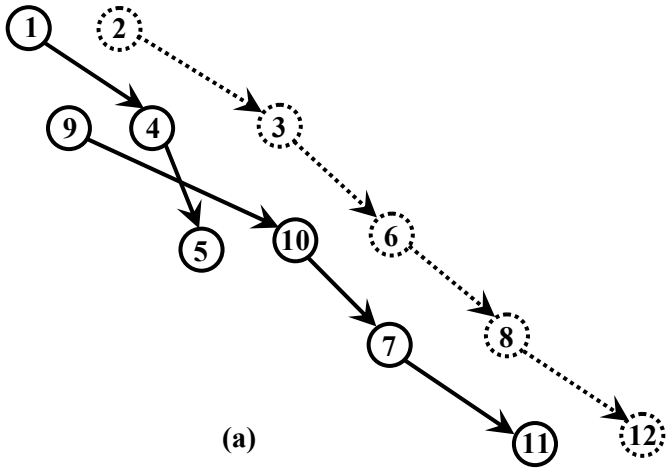
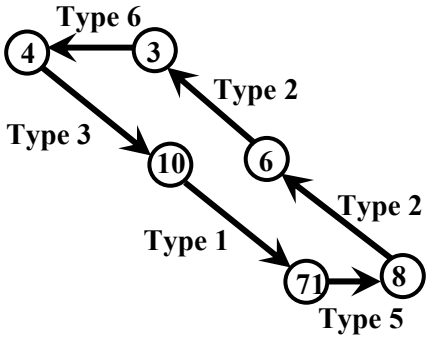
Type of Arc	Before the change in the solution graph	After the change in the solution graph	Corresponding arc in the improvement graph	Cost of the arc in the improvement graph
Type 1				$c_{ij} = (c_i^B + d_{ij}^B) - (c_i^A + d_{ij}^A)$
Type 2				$c_{ji} = (c_j^A + d_{ij}^A) - (c_j^B + d_{ij}^B)$
Type 3				$c_{il} = (c_i^B + d_{il}^B + d_{kj}^A) - (c_i^A + d_{ij}^A + d_{kl}^A)$
Type 4				$c_{li} = (c_l^A + d_{il}^A + d_{kj}^B) - (c_l^B + d_{ij}^B + d_{kl}^B)$
Type 5				$c_{ik} = (c_i^B + d_{il}^B + d_{kj}^A) - (c_i^A + d_{ij}^A + d_{kl}^B)$
Type 6				$c_{jl} = (c_j^A + d_{il}^B + d_{kj}^A) - (c_j^B + d_{ij}^B + d_{kl}^A)$

Figure 5. Different types of arcs in the A-B improvement graph.


1
2
3
4
5
6
7
8
9
10
11
12

(a)


Type 6
4
3
Type 2
Type 3
10
6
Type 2
Type 1
71
8
Type 5

(b)

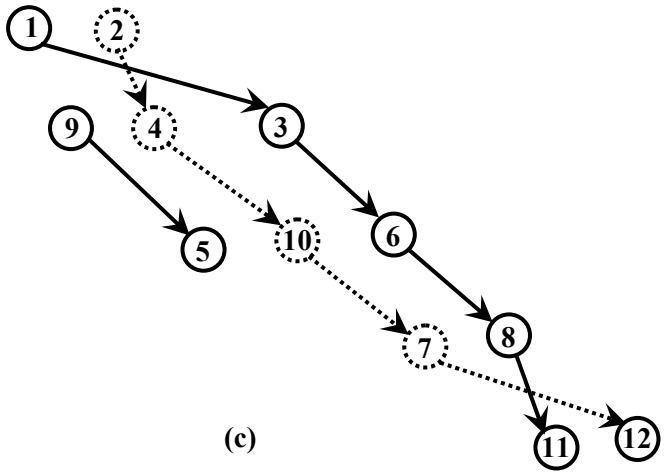


Figure 6. Valid cycles and A-B swaps.
(a) Part of the A-B solution network.
(b) A valid cycle in the improvement graph.
(c) Part of the A-B solution network when the corresponding A-B swap is performed.

Model	Local Improvement Algorithm				Tabu Search Algorithm			
	Changes in fleeting contribution (in millions)	Changes in though contribution (in millions)	Changes in total contribution (in millions)	Running time (sec.)	Changes in fleeting contribution (in millions)	Changes in though contribution (in millions)	Changes in total contribution (in millions)	Running time (sec.)
FAM without maintenance	0.55	0	0.55	3	1.77	0	1.77	144
FAM with maintenance	3.84	0	3.84	10	3.88	0	3.88	299
ctFAM without maintenance	-5.25	22.40	17.15	10	-10.00	35.20	25.20	1543
ctFAM with maintenance	-0.94	27.80	26.86	9	-2.12	31.77	29.65	380

Figure 8. Improvements obtained by the local improvement and tabu search algorithms.

Model	# of iterations	Average cost of the cycle	Average length of the cycle	Total # of flights changed	# of passes
FAM without maintenance	4	-382	3.5	13	2
FAM with maintenance	40	-263	8.0	200	3
ctFAM without maintenance	34	-956	10.4	222	3
ctFAM with maintenance	39	-1154	10.5	246	3

(a)

Model	Total # of iterations	Improving iterations	Worsening iterations	Average cost of the cycle	Average length of the cycle	# of flights changed	# of passes
FAM without maintenance	4163	1676	2487	279	4.7	41	3
FAM with maintenance	1963	619	1344	1201	5.8	215	3
ctFAM without maintenance	15354	4752	10602	1084	5.6	306	5
ctFAM with maintenance	4998	1253	3745	1471	6.2	267	4

(b)

Figure 9. Some statistics of the local search and tabu search algorithms.
(a) Local improvement algorithm; (b) Tabu search algorithm.