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

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

INFORMS Journal on Computing
Vol. 19, No. 3, Summer 2007, pp. 416–428
ISSN 1091-9856 | EISSN 1526-5528 | 07 | 1903 | 0416

DOI 10.1287/ijoc.1060.0193
© 2007 INFORMS

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

Ravindra K. Ahuja
Department of Industrial and Systems Engineering, University of Florida, Gainesville, Florida 32611,
ahuja@ufl.edu

Jon Goodstein
Information Services Division, United Airlines World Headquarters–WHQKB, Chicago, Illinois 60666

Amit Mukherjee
Automatic Data Processing, Roseland, New Jersey 07068, amit_r_mukherjee@adp.com

James B. Orlin
Sloan School of Management, Massachusetts Institute of Technology,
Cambridge, Massachusetts 02139, jorlin@mit.edu

Dushyant Sharma
Department of Industrial and Operations Engineering, University of Michigan,
Ann Arbor, Michigan 48109, dushyant@umich.edu

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, which 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. TAM is usually solved after obtaining the solution from FAM. In this sequential approach, TAM cannot change the fleeting to get a better through assignment, and FAM does not take into account the through benefits. The goal of the combined through-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 (MIP) formulation of ctFAM that is too large to be solved to even near optimality within allowable time for the data obtained by a major U.S. 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, 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 our neighborhood is very large; hence the suggested algorithm is 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 that indicate that the neighborhood search approach for ctFAM provides substantial savings over the sequential approach of solving FAM and TAM.

Key words: transportation, air; heuristics; integer programming; neighborhood search; networks
History: Accepted by Michel Gendreau, Area Editor for Heuristic Search and Learning; received July 2002; revised August 2003, March 2005; accepted March 2006. Published online in *Articles in Advance* July 20, 2007.

1. Introduction

The airline industry has been a pioneer in using IE/OR techniques to solve complex scheduling problems. Given a flight schedule, an airline needs to decide the itinerary of each aircraft and each crewmember to maximize the total revenue minus the total operating costs and meet all the operational constraints. The quality of the schedule is also measured in terms of other attributes such as reliability during operations. The entire planning problem is too large to be solved to optimality as a single optimization problem using current technology. Hence, it is typically divided into four stages (Barnhart and Talluri 1997, 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 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 advances in modeling these problems.

The next frontier in optimization of schedule planning is in solving an integrated optimization problem that considers the entire planning problem mentioned above, and includes other downstream issues that affect the overall schedule quality. Basically, the planning problem is a multicriteria optimization problem, i.e., there are many objectives that have different metrics, different priorities, and different constraints. A sequential approach has a major drawback in that the solution at each stage does not take into account subsequent stages. This results in overall suboptimal solutions. For example, if a fleet assignment is performed without considering 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.

Airlines are developing models to solve such integrated optimization problems for schedule planning by breaking down functional silos to manage planning complexity by using advances in optimization and computing. At United Airlines, two strategies are being pursued. The first is to develop explicit joint optimization models with combined objective functions, constraints, and data. These joint models are too large to be solved even to near optimality, suggesting 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 criterion, but it is hard to model in an optimization. The second strategy exploits the nature of the planning problem. There are typically many suboptimal solutions that are still close to optimal. However, these solutions can be distinct 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, could solve multicriteria scheduling problems. As a result, United Airlines initiated collaboration with the Massachusetts Institute of Technology (MIT) and the University of Florida to explore solutions that exploit the nature of the overall scheduling problem by building on the foundation for modeling the stages to develop a robust and generic methodology that could be scaled up for other criteria.

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 guarantees that our solution is no worse than that obtained by the current sequential approach, and in practice, is better.

We focus on integrating two 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).

1.1. Fleet-Assignment Model (FAM)

In FAM, planes of different fleet types are assigned to flight legs to minimize the assignment cost, subject to (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 it; and (iii) *fleet-size constraints*: 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 a MIP formulation for FAM. Clarke et al. (1996) and Subramanium et al. (1994) provide extensions to incorporate additional operational constraints related to maintenance and crew scheduling.

1.2. Through-Assignment Model (TAM)

A *through connection* is a connection between an inbound flight leg and an outbound flight leg at a station that ensures that the same plane flies both legs. Both legs in a through connection get the same flight number in the airline's flight schedule. Because a through connection allows passengers to remain onboard, passengers are willing to pay a premium, called 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 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 total 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 (see Bard and Hopperstad 1987, Barnhart et al. 1998, Gopalan and Talluri 1998, Jarrah and Reeb 1997).

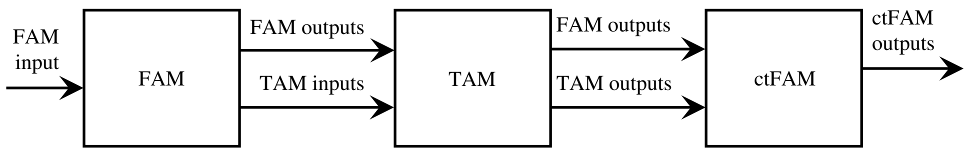


Figure 1 **Our Approach for Solving ctFAM**

1.3. Combined Through-Fleet-Assignment Model (ctFAM)

The through assignment depends on the fleet assignment in that a through connection requires that both flight legs have the same fleet type. FAM does not currently consider through benefits, and may yield fleet assignments with limited through-assignment possibilities. TAM cannot change the fleeting to get a better through assignment. In our model, ctFAM solves the integrated model and simultaneously determines fleet assignments and through connections, offering opportunities to obtain better solutions. We developed an integer programming (IP) formulation of ctFAM, which was too large to be solved even to near optimality for a major U.S. airline. We then pursued the approach outlined in Figure 1, where 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.

Neighborhood search algorithms are now widely regarded as an important tool to solve difficult combinatorial optimization problems effectively due to their intuitive appeal, flexibility, and ease of implementation, and excellent empirical results (Aarts and Lenstra 1997, Glover and Laguna 1997). We decided to pursue neighborhood search algorithms for ctFAM because of the following:

(i) They have been successful in solving a variety of large combinatorial optimization problems.

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

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

(iv) They are often flexible enough to incorporate other constraints that are difficult to model with linear constraints.

An issue of critical importance in neighborhood search is how we define the neighborhood of a solution. As in 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 trivial because the number of possible A-B swaps is exponentially large. Hence the neighborhood search algorithm using A-B swaps is a *very large-scale neighborhood* (*VLSN*) *search algorithm*, studied by Thompson and Orlin (1989), Thompson and Psaraftis (1993), and Ahuja et al. (2001a, b). Ahuja et al. (2002) present a survey of VLSN search algorithms. Several papers deal specifically with VLSN algorithms for the traveling-salesman problem, surveyed by Deineko and Woeginger (2000) and Gutin et al. (2003).

We determine the starting solution by first solving FAM followed by TAM, and successively improve it 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* changes some legs flown by fleet type A to fleet type B, changes some legs flown by fleet type B to fleet type A, and changes 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. The A-B improvement graph is constructed so that each negative-cost directed cycle satisfying some constraints defines a profitable A-B swap.

The neighborhood search algorithm constructs the A-B improvement graph and solves an IP to identify a negative-cost constrained directed cycle. This cycle yields a new fleet and through assignment with 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.

The solutions obtained by our algorithms resulted in annual savings of \$5 million to \$25 million on the data provided by United Airlines. These results suggest that neighborhood search is useful.

We generalize Talluri's (1996) concept of an A-B improvement graph to incorporate through constraints. Our approach, when specialized to FAM, provides a neighborhood that contains Talluri's neighborhood 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, and our neighborhood structure does not make it. Indeed, it does not even depend upon the existence of arrival and departure banks.

In Section 2 we present an IP formulation for ctFAM. Section 3 develops our neighborhood search algorithms for ctFAM. We give our computational results in Section 4. Section 5 gives conclusions and avenues of future research.

2. An Integer-Programming Formulation of ctFAM

In this section, we present a linear integer-programming (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.

2.1. Input Data

L: The set of all flight legs that 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.

$dep\text{-}time(i)$: The departure time for flight leg i.

$arr\text{-}time(i)$: The arrival time for flight leg i. Let $arr\text{-}time(i)$ be 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; in practice, the turn time also depends on the fleet type but for simplicity of notation, we assume it to be independent of the fleet type). Thus, the plane released from flight i can be assigned to any flight j with $dep\text{-}time(j) \geq arr\text{-}time(i)$ at the same city.

$dep\text{-}city(i)$: The departure city for flight leg i.

$arr\text{-}city(i)$: The arrival city for flight leg i.

c_i^f: The cost incurred in assigning fleet type f to flight leg i, being the negative of the estimated contribution from the assignment of f to i. The contribution is determined by estimated revenue, spill cost, and operating costs such as fuel, crew, aircraft maintenance, etc.

d_{ij}^f: The cost incurred in connecting flight leg i with the flight leg j provided $arr\text{-}city(i) = dep\text{-}city(j)$ and both legs are flown by fleet type f. It is the negative of the estimated marginal increase in revenue when the pair of flights i and j is declared a through. 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, $count\text{-}time \neq arr\text{-}time(i)$ or $dep\text{-}time(i)$ for any $i \in L$. We will assume here that *count-time* is midnight.

2.2. Connection Network

We now explain how to construct the connection network, which will be the basis of our IP as well as our neighborhood search algorithm for ctFAM. Let the connection network $G = (N, E)$ where N is the node set and E is the arc set. The node set $N = \{i\colon i \in L\}$ is obtained by defining a node for each flight leg $i \in L$, and the arc set $E = \{(i, j)\colon arr\text{-}city(i) = dep\text{-}city(j)\}$ consists of all possible connections between inbound and outbound flight legs. Obviously, a 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. An example of connection network is 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 use the following additional notation related to the connection network: $I(i) = \{(j, i) \in E\colon j \in N\}$, $O(i) = \{(i, j) \in E\colon j \in N\}$, $S = \{(i, j) \in E\colon arr\text{-}time(i) < count\text{-}time < arr\text{-}time(j)\} \cup \{(i, j) \in E\colon dep\text{-}time(i) < count\text{-}time < arr\text{-}time(i)\}$, where the inequalities are based on circular 24-hour time.

Let $I(i)$ be the set of incoming arcs at node i in the connection network, let $O(i)$ be the set of outgoing arcs, and let S be the set of arcs in the connection network that cross the *count-time*. We call the arcs

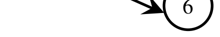


Figure 2 Part of the Connection Network at a City with Inbound Flights 1, 2, and 3, and Outbound Flights 4, 5, and 6

in S *overnighting* arcs, which include connection arcs that cross the *count-time*, and also those whose arrival flights are in the air at *count-time*.

2.3. Decision Variables

We define two sets of IP decision variables:

$y_i^f = 1$ if the flight leg i is assigned fleet type f, and 0 otherwise.

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

The IP of ctFAM is

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

$$\text{subject to} \sum_{f\in F} y_i^f = 1, \quad \text{for all } i \in N \tag{1b}$$

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

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

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

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

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

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 the Online Supplement to this paper on the journal's website. We also explain how our algorithm needs to be modified to account for these constraints in Appendix II in the Online Supplement along with other implementation details.

Though ctFAM can be formulated as an IP, it is too large to be solved even to near optimality (using current LP technology) for the national network of a large U.S. airline. In the data instances supplied by United Airlines for their daily network, there were 13 fleet types, and more than 1,600 flight legs and 13,900 through connections. The resulting IP had approximately 100,000 integer variables and 18,000 constraints. We could not solve problems of this magnitude using commercial IP solvers. We then focused on neighborhood search algorithms to solve ctFAM. Additional reasons for considering neighborhood search algorithms are 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 $\mathbf{N}(x, y)$ of neighboring solutions. Let $c(x, y)$ denote the objective function (1a) of a solution (x, y) in the IP in Section 2. 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 $(x', y') \in \mathbf{N}(x, y)$
    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;
```

Though the neighborhood search algorithm stated above always replaces (x, y) by an improved neighbor (x', y'), there exist variants that occasionally replace (x, y) by worse neighbors including simulated annealing and tabu search. (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 $\mathbf{N}(x, y)$ with respect to the solution (x, y); and (iii) searching the neighborhood $\mathbf{N}(x, y)$ to identify an improved solution.

3.1. 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 FAM, minimizing the first double summation in (1a), which is the negative of the contribution from assignment of fleet types to the flight legs. The solution of FAM gives us a fleet assignment assigning a plane type to each flight leg. We next use TAM to generate a set of connections, minimizing the second double summation in (1a), which is the negative of the estimated through revenue obtained by a through assignment. We solve TAM 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.

3.2. 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 is $N(S^{AB}(x, y)) = \{i \in N\colon y_i^A = 1 \text{ or } y_i^B = 1\}$ and arc set is $E(S^{AB}(x, y)) = \{(i, j) \in E\colon x_{ij}^A = 1 \text{ or } x_{ij}^B = 1\}$.

In other words, the A-B solution graph $S^{AB}(x, y)$ is the subgraph of G whose node set is composed of the flight legs assigned fleet types A and B in the solution (x, y), and the arc set is composed of the connections between those flight legs. We 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 refer to an arc in the *A-B* solution graph as an *A-arc* if $x_{ij}^A = 1$ and *B-arc* if $x_{ij}^B = 1$.

3.3. 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. Obtaining an A-B neighbor is called an *A-B swap*. Figure 3(a) shows part of the solution graph $S^{AB}(x, y)$ and Figure 3(b) shows the same part after the A-B swap. We show A-nodes and A-arcs using solid lines, and B-nodes and B-arcs using dashed lines. 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 only with the same fleet type.

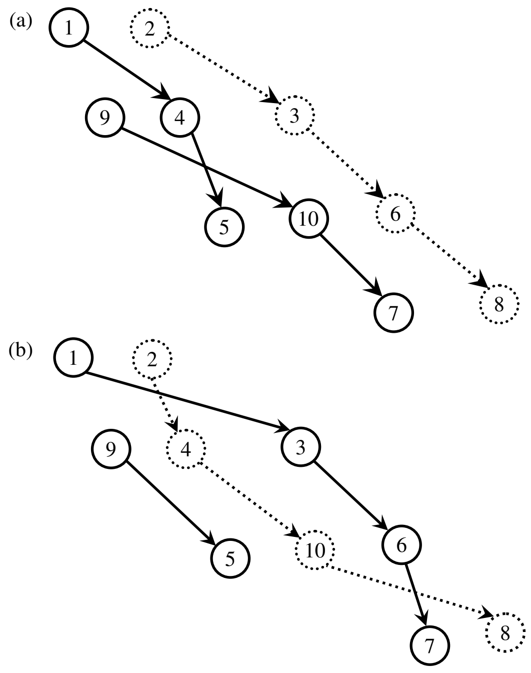


Figure 3 Illustrating an A-B Swap. Solution Graph Before the A-B Swap (a) and After the A-B Swap (b)

While defining A-B swaps we maintain fleet-size constraints for fleet types A and B. An A-B swap may result in changing the number of aircraft of fleet type A or B overnighting locally at some stations while satisfying the fleet-size constraints for fleet types A and B. Figure 4(a) shows part of the A-B swap where the number of planes of a particular type overnighting at a station can change. 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 P.M. and 4 P.M., respectively, and the flights 3 and 4 depart at 5 P.M. and 3 P.M., 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 P.M. and connects to flight 4, which departs at 3 P.M. 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

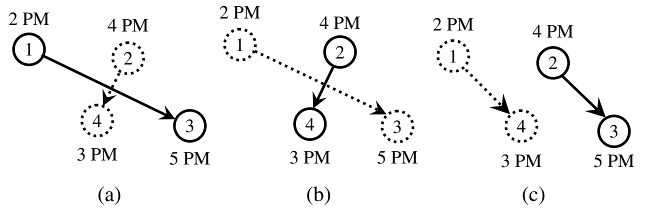


Figure 4 Effect of Swaps on the Number of Planes Used

Notes. (a) Four flights and two connections. (b) A possible swap that increases the number of planes used for type A and decreases the number of planes used for type B. (c) A possible swap that decreases the number of planes used for type B.

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. 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 neither of the new connection arcs $(1, 4)$ and $(2, 3)$ is an overnight arc. This swap will reduce the number of planes used for type B by one. Our neighborhood search algorithm allows all such swaps.

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 IP. 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 an *A-B* improvement graph, which allows us to identify profitable *A-B* swaps efficiently over a structured subset of the A-B neighborhood.

3.4. A-B Improvement Graph

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 so that an improving cycle leads to a new solution with the above cycle-based property.

We first illustrate the simplest type of swap before moving to the more complex swaps. 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 more simply 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 satisfies 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 so 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. The changes corresponding to an arc may result in locally changing the count of planes of type A or B overnighting at a station, as observed in Figure 4. For each arc (i, j), we compute a_{ij} and b_{ij} as the change in the number of planes of type A and B, respectively, overnighting at the station associated with arc (i, j) when the changes corresponding to the arc are performed. The station associated with arc (i, j) is the city *arr-city*(i) if i is an 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, where we show an A-node or an A-arc using solid lines, and a B-node or B-arc using dashed lines. Detailed explanation of these arcs is given next.

3.4.1. Type 1 Arcs. Consider an arc (i, j) in the A-B solution graph which is an A-arc. We instroduce the arc (i, j) in the improvement graph that 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 c_{ij} of the arc (i, j) 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. 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. If $(i, j) \notin S$ then $a_{ij} = b_{ij} = 0$, otherwise $a_{ij} = -1$ and $b_{ij} = 1$.

3.4.2. Type 2 Arcs. A type 2 arc (j, i) is introduced in the improvement graph for each B-arc (i, j) in the

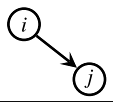
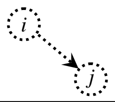
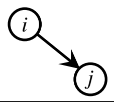
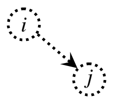
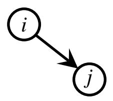
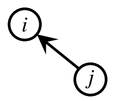
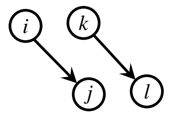
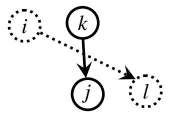
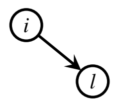
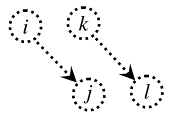
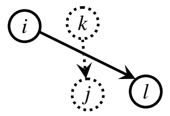
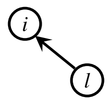
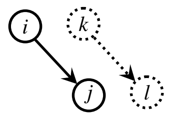
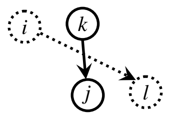
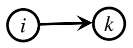
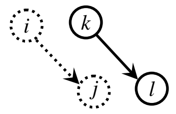
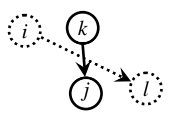
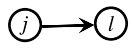
Type of arc	Before the change in the solution graph	After the change in the solution graph	Arc in the improvement graph	Cost of the arc in the improvement graph
Type 1				$c_{ij} = (c_i^B + d_{ij}^B) - (c_i^A + d_{ij}^A)$
Type 2				$c_{ji} = (c_j^A + d_{ij}^A) - (c_j^B + d_{ij}^B)$
Type 3				$c_{il}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

A-B solution graph. 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. Contrary to the case of type 1 arcs, we introduce the arc (j, i) instead of arc (i, j). The arcs are reversed 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). If $(i, j) \notin S$ then $a_{ij} = b_{ij} = 0$, otherwise $a_{ij} = 1$ and $b_{ij} = -1$.

3.4.3. 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 *arr-city*(i) = *arr-city*(k). 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. The value of $b_{il} = 1$ if $(i, l) \in S$ and 0 otherwise. The value of a_{il} can be computed by checking membership of (i, j), (k, l), and (k, j) in S.

3.4.4. Type 4 Arcs. We introduce a type 4 arc (l, i) in the improvement graph for every pair of B-arcs (k, l) and (i, j) in the A-B solution graph such that *arr-city*(i) = *arr-city*(k). The cost of the arc (l, i) includes the costs of changing fleet type of flight j and l 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.

3.4.5. 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 *arr-city*(i) = *arr-city*(k). 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. The value of a_{ik} is 1 if $(k, j) \in S$ and $(i, j) \notin S$, is –1 if $(k, j) \notin S$ and $(i, j) \in S$, and is 0 otherwise. The value of b_{ik} can be computed similarly by checking membership of (i, l) and (k, l) in S.

3.4.6. 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 *arr-city*(i) = *arr-city*(k). 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. The values of a_{jl} and b_{jl} can be computed similarly to type 5 arcs.

We identify A-B swaps by defining valid cycles. 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 $mate(i) = k$.

3.4.7. Valid Cycles. *A directed cycle W in the A-B improvement graph is said to be a valid cycle if* $\sum_{(i,j)\in W} a_{ij} \le 0$, $\sum_{(i,j)\in W} b_{ij} \le 0$, and it satisfies $mate(i) \notin W$ unless $(i, mate(i)) \in W$ for every node $i \in W$.

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 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 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 changes only the fleeting. The next arc $(7, 8)$ in the cycle is

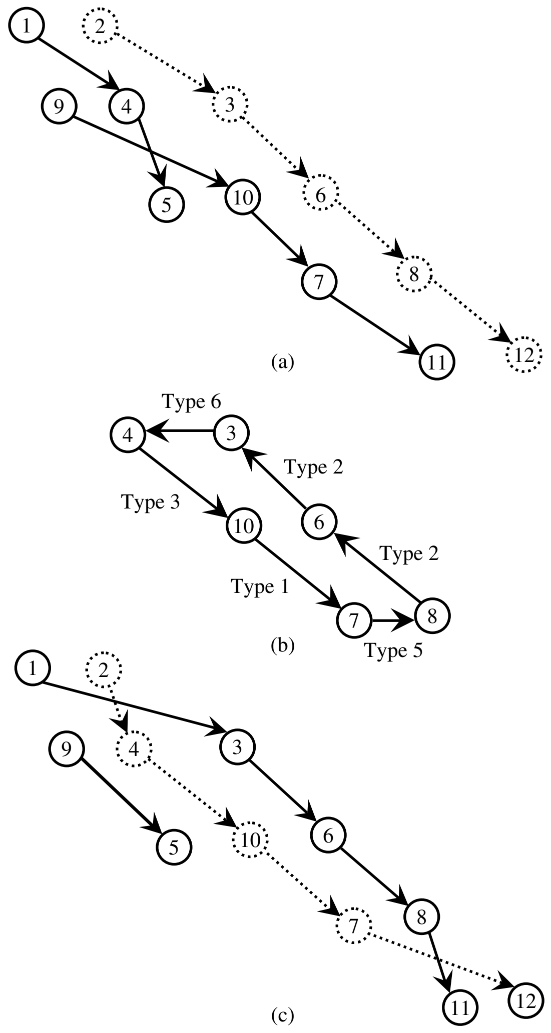


Figure 6 Valid Cycles and A-B Swaps
Notes. (a) Part of the A-B solution network. (b) A valid cycle in the improvement graph. (c) Part of the A-B solution network when the corresponding A-B swap is performed.

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. Any A-B swap results in a solution satisfying the constraint (1b) because 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 constraint (1e) is also satisfied because the changes corresponding to each arc in the A-B improvement graph ensure that the fleet size constraints (1e) are satisfied. We now show that 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 prove in cases that this property is satisfied after the A-B swap. 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 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.) $\square$

3.5. Identifying A-B Swaps

We showed in Section 3.4 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 III in the Online Supplement). 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 IP.

We first introduce some notation related to the IP. Let $N' = N(G^{AB}(x, y))$ be the set of nodes and $E' = E(G^{AB}(x, y))$ be 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. The IP formulation is

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

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

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

$$\sum_{(i,j)\in E'} a_{ij} w_{ij} \le 0 \tag{2d}$$

$$\sum_{(i,j)\in E'} b_{ij} w_{ij} \le 0 \tag{2e}$$

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

Constraints (2b) and (2f) imply that the solution is a 0-1 circulation, which can be decomposed into unit flows along directed cycles. Constraints (2c) ensure that the flow passing through each node i plus the flow passing through the node $\mathit{mate}(i)$ is at most 1, which implies that the resulting flow will not pass through both the nodes i and $\mathit{mate}(i)$. An exception to this rule occurs when flow takes place over the arc $(i, \mathit{mate}(i))$ in which case both the nodes i and $\mathit{mate}(i)$ can be visited. It is easy to see that a feasible solution of (2) gives a set of valid cycles. Constraints (2d) and (2e) ensure that the number of planes of type A and B used in the solution does not increase. 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 minimum total cost. Using flow decomposition (see, for example, Ahuja

```
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**

et al. 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.

4. Computational Testing

We programmed our algorithms in C on a Sun workstation with a 1.2 GHz Superscalar SPARC V9 processor and 8 GB of RAM. We tested our algorithms on the data provided by United Airlines.

We tested our local improvement and tabu search algorithms on two types of problems using (i) ctFAM without maintenance constraints; and (ii) ctFAM with maintenance constraints. We were provided one instance for each problem type. We generated four additional (smaller) instances of each type for computational testing. These instances are generated using the given instances. We generated the additional instances as follows. We first solved the FAM model for each original instance to find a feasible fleet assignment. To generate each smaller instance of a problem type, a subset of the fleet types was chosen and all the flight legs that are assigned to one of the chosen fleet types were selected. The set of possible through connections was chosen as the subset of the given through connections that were possible for the selected flight legs. Our choice of flight legs and possible through connections ensures that the resulting instance has a feasible solution. In the case of instances for ctFAM with maintenance constraints, we also need to decide which of the additional three constraints must be satisfied in the instance. We identified an appropriate subset of the original additional constraints for each of the smaller instances by trial-and-error to guarantee that the instances are feasible. Lastly, to distinguish the objective function of the derived instances from the given instances, we perturbed the fleet-assignment costs in the new instance by adding a uniform random variable to each value. The uniform variable was chosen to have mean zero and range equal to 1% of the original cost value on either side of zero. Table 1 gives the important characteristics of the ten problem instances tested. The instances 1a–1e are instances of ctFAM without maintenance constraints, and 2a–2e are instances with maintenance constraints. The original instances provided to us are 1e and 2e.

Table 1 **Characteristics of Problem Instances**

Problem	Flight legs	Through connections	Fleet types	Number of aircraft
1a	359	808	7	86
1b	683	3,017	8	150
1c	942	5,519	9	215
1d	1,268	10,169	11	276
1e	1,605	13,915	13	354
2a	373	916	7	87
2b	696	3,296	8	151
2c	954	5,557	9	216
2d	1,269	10,097	11	277
2e	1,609	13,964	13	355

Notes. (1a–1e) Instances of the ctFAM without maintenance constraints. (2a–2e) Instances of the ctFAM with maintenance constraints.

We obtained the starting solutions for the neighborhood search algorithms by first solving FAM to optimality and then applying TAM. Our objective in the neighborhood search was to maximize the total fleet assignment and through contribution, which is the negative of the objective in the integer program. Table 2 gives the changes in fleet and through contributions by the use of neighborhood search on the solution obtained by sequential application of FAM and TAM. We also provide the maximum possible improvement in the overall objective, which is obtained by solving the IP for ctFAM to optimality. The improvements obtained are reported on an annual basis. The following additional conclusions can be drawn from the table:

- Overall there is significant suboptimization resulting from the current sequential approach.

Table 2 **Results from Local Improvement, Tabu Search, and MIP Algorithms**

	Changes in fleeting contribution (millions)			Changes in through contribution (millions)			Changes in total contribution (millions)			Running time (seconds)		
Problem	LS	TS	MIP	LS	TS	MIP	LS	TS	MIP	LS	TS	MIP
1a	−0.004	−0.004	−0.004	0.324	0.324	0.324	0.32	0.32	0.32	1	1	1
1b	−0.07	−2.56	−1.75	1.02	3.94	3.65	0.95	1.38	1.90	2	34	19
1c	−1.47	−5.13	−9.10	10.14	18.48	24.31	8.67	13.35	15.22	6	146	92
1d	−5.19	−7.57	−19.43	25.38	36.06	55.86	20.19	28.49	36.44	17	604	52,319
1e	−5.49	−12.07	—	30.82	49.78	—	25.22	37.71	60.69*	27	1,148	5×10^5
2a	−0.45	−0.55	−1.05	1.85	2.13	2.76	1.40	1.58	1.72	1	2	2
2b	−0.34	−0.72	−1.82	2.32	3.87	5.42	1.98	3.15	3.60	2	9	133
2c	−0.78	−3.98	−10.62	9.57	17.15	25.66	8.79	13.17	15.03	5	17	370
2d	−3.27	−7.44	—	22.58	36.26	—	19.31	27.82	51.70*	13	187	5×10^5
2e	−3.84	−11.12	—	29.60	47.67	—	25.86	36.55	71.45*	24	316	5×10^5

*Best upper bound found within the allotted time limit.

Table 3 **Some Statistics of the Local Search and Tabu Search Algorithms: Local Improvement Algorithm**

Problem	No. of improving iterations	Average cost of the cycle	Average length of the cycle	Total no. of flights changed	No. of passes
1a	1	−877	8	8	2
1b	2	−824	6	12	2
1c	5	−3,054	4.6	23	2
1d	21	−2,042	10.6	150	3
1e	29	−1,935	11.8	242	3
2a	1	−3,835	2	2	2
2b	2	−2,712	2	4	2
2c	5	−3,178	5.2	22	2
2d	20	−1,968	7.9	142	3
2e	27	−2,024	9.7	207	3

• The time requirement of both the search algorithms scales well with problem size, whereas that of the IP approach does not.

• The local improvement algorithm is efficient and terminates quickly. The neighborhood search algorithms found the possible improvements quickly. This suggests that neighborhood search algorithms can be used as a supplement to the IP.

We performed some additional computational results to assess the behavior of our algorithms. Tables 3 and 4 show the statistics we noted for our two models and the two algorithms.

5. Conclusions

We have studied the combined through-fleet-assignment model (ctFAM), which integrates the fleet-assignment model (FAM) and the through-assignment model (TAM). We give an IP formulation of ctFAM that, unfortunately, is too large to be solved even to near optimality using state-of-art commercial IP solvers. We propose a swap-based neighborhood search algorithm for ctFAM that swaps 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 IP 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 and tabu search algorithms obtained substantial improvements in savings and computational times.

Table 4 **Some Statistics of the Local Search and Tabu Search Algorithms: Tabu Search Algorithm**

Problem	Total no. of iterations	Worsening iterations	Improving iterations	Average cost of the cycle	Average length of the cycle	No. of flights changed	No. of passes
1a	37	36	1	7,427	5.1	8	2
1b	1,240	866	374	1,695	4.8	28	2
1c	2,149	1,410	739	1,350	5.5	78	3
1d	8,515	5,597	2,918	936	5.4	183	4
1e	17,336	11,856	5,480	933	5.8	353	5
2a	15	13	2	2,297	3.5	5	2
2b	55	51	4	1,187	4.5	8	2
2c	99	91	8	1,038	4.1	29	2
2d	3,348	2,383	965	1,298	5.2	165	3
2e	5,117	3,853	1,264	1,265	5.7	262	4

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 multicriteria optimization problems by incorporating other downstream criteria in schedule planning.

A comprehensive framework has been developed for airlines to model and solve advanced planning problems.

Acknowledgments
The authors gratefully acknowledge support through NSF Grants DMI-9900087 and DMI-0217359 (at the University of Florida) and DMI-9820998 and DMI-0217123 (at MIT) and a grant from United Airlines. The authors also thank Raj Sivakumar and Ram Narsimhan at United Airlines for their help and support during the course of this project.

References
Aarts, E., J. K. Lenstra, eds. 1997. *Local Search in Combinatorial Optimization*. John Wiley and Sons, New York.

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

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

Ahuja, R. K., J. B. Orlin, D. Sharma. 2001a. A composite neighborhood search algorithm for the capacitated minimum spanning tree problem. *Oper. Res. Lett.* **31** 185–194.

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

Ahuja, R. K., O. Ergun, J. B. Orlin, A. P. Punnen. 2002. A survey of very large-scale neighborhood search techniques. *Discrete Appl. Math.* **123** 75–102.

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

Barnhart, C., K. T. Talluri. 1997. Airlines operations research. A. McGarity, C. ReVellepp, eds. *Design and Operation of Civil and Environmental Engineering Systems*. Wiley Interscience, New York, 435–469.

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

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

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

Deineko, V. G., G. J. Woeginger. 2000. A study of exponential neighborhoods for the traveling salesman problem and for the quadratic assignment problem. *Math. Programming* **87** 519–542.

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

Gopalan, R., K. T. Talluri. 1998. Mathematical models in airline schedule planning: A survey. *Ann. Oper. Res.* **76** 155–185.

Gutin, G., A. Yeo, A. Zverovitch. 2003. Exponential neighborhoods and domination analysis for the TSP. G. Gutin, A. P. Punnen, eds. *Traveling Salesman Problem and Its Variations*. Kluwer, Boston, MA.

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

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

Subramanium, R., R. P. Scheff, Jr., J. D. Quillinan, D. S. Wiper, 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 Sci.* **31** 237–248.

Thompson, P. M., J. B. Orlin. 1989. The theory of cyclic transfers. Working Paper OR 200-89, Operations Research Center, MIT, Cambridge, MA.

Thompson, P. M., H. N. Psaraftis. 1993. Cyclic transfer algorithms for multivehicle routing and scheduling problems. *Oper. Res.* **41** 935–946.