from *SIAM News*, Volume 35, Number 9, November 2002

Very Large-Scale Neighborhood Search in Airline Fleet Scheduling

By Ravindra K. Ahuja and James B. Orlin

Many discrete optimization problems of practical interest cannot be solved to optimality in the available time. A practical approach to these problems is to use heuristic algorithms, which do not guarantee the optimality of the solution but which, for many problems, can find nearly optimal solutions within a reasonable amount of computational time.

The literature on heuristic algorithms often distinguishes between two broad classes: *constructive algorithms* and *improvement algorithms*. A constructive algorithm builds a solution from scratch by assigning values to one or more decision variables at a time. An improvement algorithm starts with a feasible solution and iteratively tries to improve it.

Neighborhood search algorithms, also called *local search algorithms*, are a wide class of improvement algorithms that at each iteration search the "neighborhood" of the current solution to find an improved solution. A neighborhood search algorithm for a discrete optimization problem **P** (where we wish to minimize an objective function over a finite discrete set of objects) starts with a feasible solution x^1 of the problem. For each feasible solution x, an associated neighborhood of x, denoted $\mathbf{N}(x)$, is a set of feasible solutions that can be obtained by perturbing the solution x using some prespecified scheme. The elements of $\mathbf{N}(x)$ are called *neighbors* of x.

The neighborhood search iteratively obtains a sequence $x^1, x^2, x^3, \ldots$ of feasible solutions. At the kth iteration, the algorithm determines a solution x^{k+1} with a lower objective function value than x^k, if one exists. The algorithm terminates when it finds a solution that is at least as good as any of its neighbors; such a solution is called a *locally optimal solution*. Typically, multiple runs of the neighborhood search algorithm are performed with different starting solutions, and the best locally optimal solution is selected. Figure 1 illustrates one run of the neighborhood search algorithm. A comprehensive discussion of neighborhood search algorithms can be found in [1].

A critical issue in the design of a neighborhood search algorithm is the choice of the neighborhood structure—that is, the manner in which the neighborhood $\mathbf{N}(x)$ is defined for the solution x. This choice largely determines whether the solutions obtained will be highly accurate or will have poor local optima. As a rule of thumb, the larger the neighborhood, the better will be the quality of the locally optimal solutions, and the greater the accuracy of the final solution. At the same time, searching larger neighborhoods requires more time at each iteration. Because of the many runs of a neighborhood search algorithm usually performed, longer execution times per run lead to fewer runs within a specified time. For this reason, a larger neighborhood can produce a more effective heuristic algorithm only if the larger neighborhood can be searched in a very efficient manner.

Most of the neighborhood search algorithms in the literature use small neighborhoods, explicitly enumerating all neighbors in the search. The algorithms we describe here search very large neighborhoods, and we refer to them as *very large-scale neighborhood (VLSN) search algorithms*. VLSN algorithms open up myriad possibilities for neighborhood search. For some problems, we can search a large neighborhood in a time roughly comparable to that needed for a small neighborhood, but with far better solution quality. A survey of VLSN search techniques can be found in [2]. We illustrate the VLSN search technique here for airline scheduling, a problem for which a small neighborhood search is not likely to be effective [3].

```
algorithm neighborhood search;
begin
        obtain an initial feasible solution x^1 of P;
        set k := 1;
        while there is a neighbor x ∈ N(x) with c(x) < c(x^k) do
        begin
                set k := k+1;
                set x^k := x;
        end;
        output x^k which is a locally optimal solution;
end;
```

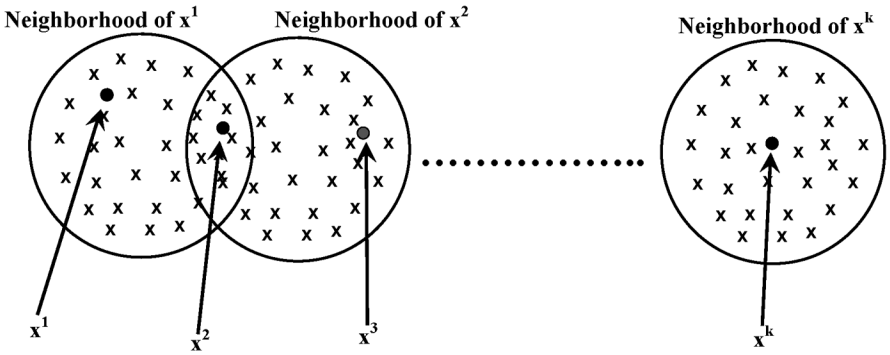

Figure 1. *A neighborhood search algorithm.*

Airline Fleet Scheduling

The airline industry has been a pioneer in the use of operations research techniques. Given a flight schedule, an airline's schedule-planning group needs to decide on an itinerary for each aircraft and each crewmember that maximizes total revenue

minus total operating costs and, at the same time, satisfies all operational constraints. The entire planning problem is clearly too large to be solved to optimality as a single optimization problem.

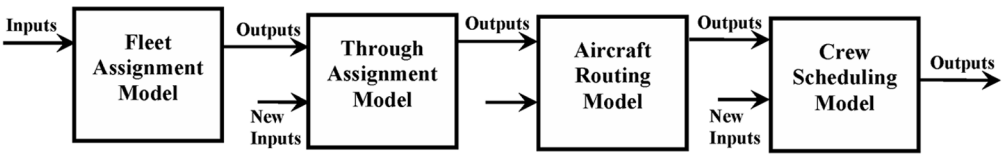

Figure 2. *Various stages in the generation of an airline schedule.*

At United Airlines, the planning problem is divided into four stages [4,5]: (i) *fleet assignment*, (ii) *through assignment*, (iii) *aircraft routing*, and (iv) *crew scheduling*. These problems are solved as a sequence of four optimization problems, with the optimal solution of one problem becoming the input for the following problem (see Figure 2). Together, these four problems model much of the cost of running the airline. A relatively small improvement in the solutions for these models can result in considerable improvement in the bottom line.

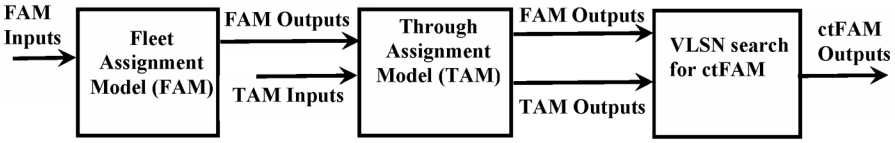

Figure 3. *Our approach for solving ctFAM.*

The sequential approach has a major drawback: The solution at each stage does not take into account considerations that will be important in the subsequent stages. This results in suboptimal overall solutions. For example, fleet assignment is performed without considerations of optimizing crew scheduling. If the fleet assignment model were to incorporate crew issues, it would be likely to provide a better starting point to the crew-scheduling optimization, resulting in overall economic benefit to the airline.

The ultimate optimization model in airline schedule planning is an integrated model that addresses the entire planning problem mentioned earlier. Because this is an immensely difficult mathematical programming problem, airlines are moving toward this goal by integrating two models into a single model at a time. We describe here an approach that integrates the fleet assignment and the through assignment models into a single model, called the *combined through–fleet assignment model*.

In the fleet assignment model, planes of different types (e.g., Boeing 767, Airbus 310) are assigned to flight legs (e.g., Boston to Chicago, departing at 8 AM) in the airline's schedule in a way that minimizes the assignment cost. The decision variable in the fleet assignment model, $x(l,k)$, defined for each flight leg k and each fleet type l, takes value 1 if we assign fleet type l to flight leg k and 0 otherwise. Any feasible fleet assignment solution must satisfy the following constraints: (i) *covering constraints*—each flight leg l must be assigned exactly one plane, i.e., $x(l,k)$ is 1 for exactly one fleet type k; (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 constraints*—for each fleet type, the number of planes used must not exceed the number of planes available. The fleet assignment model can be formulated as an integer programming model. With current commercial software, the fleet assignment model can be solved in a few hours on a personal computer.

For the through assignment model, we define a *through connection*: a connection between an inbound flight leg, say from City X to City Y, and an outbound flight leg from the same city, say from City Y to City Z, satisfying the constraint that both legs are flown by the same plane. A through connection shows up as a new flight leg, City X to City Z with one stop in City Y, in the airline's published schedule. Passengers traveling from City X to City Z prefer a through connection over a non-through connection—they can remain on board at City Y, instead of changing gates, and they have no connecting flight to miss. The premium that passengers are willing to pay for through connections is called the *through benefit*. The through assignment model takes as input a list of pairs of flight legs that can make through connections, with the corresponding through benefits, and identifies a set of most profitable through connections.

The system now in place at United Airlines begins with the fleet assignment model, which is solved by integer programming techniques. The output of the fleet assignment model becomes an input of the through assignment model, which thus creates through connections between flight legs only if they have been assigned the same fleet types in the fleet assignment model. The through assignment model is also solved by integer programming techniques.

Because the fleet assignment model does not take into account the through benefits, its fleet assignments have limited through assignment possibilities. In the combined through–fleet assignment model (ctFAM), we solve the integrated model and simultaneously determine the fleet assignments and through connections. The integrated model offers opportunities for solutions better than those obtained with the current sequential approach.

Our initial approach was to develop an integer programming formulation for the ctFAM. However, this formulation was too large to be solved to optimality or near-optimality for the national network of United Airlines. We then pursued the approach outlined in Figure 3. The new approach starts with the current solution to the fleet and through assignments (as in the first two boxes), and then successively improves it using our very large-scale neighborhood search algorithm. This approach produces a solution at least as good as, and in practice better than, United's current approach.

Our VLSN search algorithm generalizes the approach of Talluri [6]. 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 some legs flown by fleet type A to fleet type B, changing some legs flown by fleet type B to fleet type A, and then changing through connections as needed so that all constraints remain satisfied. We illustrate an A–B swap in Figure 4. In the figure, we represent each flight by a node, and connections between flights by arcs. Nodes corresponding to fleet type A are depicted by solid lines, those corresponding to fleet type B by dotted lines. Only nodes representing the same fleet type are connected.

An A–B swap is *profitable* if performing it decreases the total cost of the solution, which in our case includes fleet assignment costs and through benefits. Identifying a profitable A–B swap requires an efficient search technique, because the number of possible A–B swaps can be very large. We developed a method that uses the concept of an *A–B improvement graph,* G^{AB}, which satisfies the property that each negative cost directed cycle in the graph satisfying some additional constraints defines a profitable A–B swap.

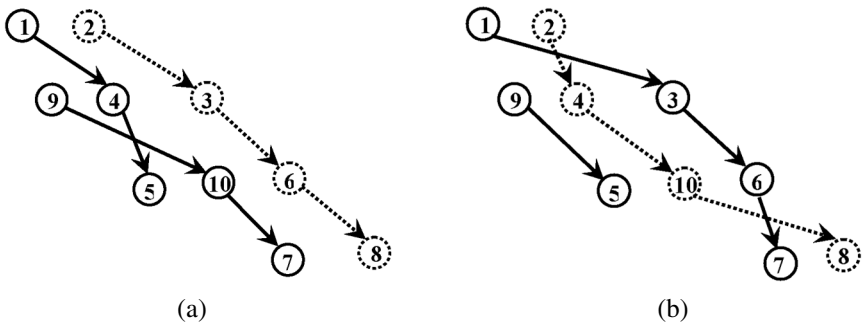

Figure 4. *Part of the solution (a) before and (b) after an A–B swap.*

Profitable A–B swaps are called *valid cycles.* Because G^{AB} can contain an exponentially large number of valid cycles, the size of our neighborhood is indeed very large. We formulated the problem of identifying a valid cycle in G^{AB} as an integer programming problem and solved it using CPLEX, a commercial integer programming solver. This valid cycle yields a profitable A–B swap with respect to the given solution of ctFAM; performing this swap gives a new fleet and through assignment with a lower cost. We update the A–B improvement graph with respect to the new solution, repeating the process until there is no constrained negative cycle in the improvement graph with respect to the current solution. At this point, we consider a different combination of fleet types. There are 13 fleet types, so we consider $13 \times 12 = 154$ different combinations of fleet types for possible improvement. We terminate when for every pair of fleet types A and B, the A–B improvement graph contains no valid cycle. The algorithm is shown in Figure 5. Details on constructing the A–B improvement graph and on finding a valid cycle can be found in [3].

```
algorithm ctFAM neighborhood search;
begin
    determine the optimal fleet assignment;
    determine the optimal through connections;
    repeat
        for each pair of the fleet types A and B do
        begin
            construct the A–B improvement graph G^AB;
            while the G^AB contains a valid cycle do
            begin
                determine a valid cycle W in G^AB;
                perform A–B swaps corresponding to W;
                update the improvement graph G^AB;
            end;
        end;
    until G^AB contains no valid cycle
            for any pair of fleet types A and B;
end;
```

Figure 5. *The neighborhood search algorithm for ctFAM.*

Working with researchers at United Airlines, we developed and implemented the VLSN search algorithm for ctFAM. For the data provided by United for its national network, the algorithm terminated in about 6 seconds, with an estimated annual saving of more than $25 million compared with the system now in use. The VLSN search algorithm is currently being put into production.

Summary

VLSN search opens up many possibilities in the area of heuristic search. It can be advantageous for several reasons. Most immediately, for some problems searching a large neighborhood can be more effective than searching smaller neighborhoods. In some cases there simply is no small neighborhood that can be used for a neighborhood search. The VLSN search for airline fleet scheduling described in this article is one such case—no existing natural small neighborhood could lead to improved solutions.

VLSN search adds flexibility and power to existing search techniques, especially when used in conjunction with those techniques. The VLSN search can also take advantage of subproblems that may be easier to solve.

Human schedulers sometimes use computer algorithms as part of an iterative process to find improved solutions. In this case, the human is doing the scheduling, and the computer is offering suggested changes to the current solution. Neighborhood search is required to keep the solution stable from iteration to iteration, with no solution differing markedly from its predecessor. Finally, from the viewpoint of applied mathematics, VLSN search has intrinsic interest; it opens up a domain of interesting mathematical problems for which answers are needed.

Acknowledgments

The work described in this article was funded in part by the National Science Foundation and United Airlines. We thank Dushyant Sharma (MIT) for his help in this project. For the research with United Airlines, we also acknowledge our collaborators and co-authors there: Jon Goodstein and Amit Mukherjee.

References

[1] E. Aarts and J.K. Lenstra, *Local Search in Combinatorial Optimization*, John Wiley, New York, 1997.

[2] R.K. Ahuja, O. Ergun, J.B. Orlin, and A.P. Punnen, *A survey of very large scale neighborhood search techniques*, to appear in Discrete Appl. Math., 2002.

[3] R.K. Ahuja, J. Goodstein, A. Mukherjee, J.B. Orlin, and D. Sharma, *A very large-scale neighborhood search algorithm for the combined through–fleet assignment model*, Working Paper, Department of Industrial and Systems Engineering, University of Florida, Gainesville, Florida,

submitted for publication, 2001.

[4] C. Barnhart and K.T. Talluri, *Airlines operations research*, in *Design and Operation of Civil and Environmental Engineering Systems*, A. McGarity and C. ReVelle, eds., Wiley InterScience, New York, 1997, 435–469.

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

[6] K.T. Talluri, *Swapping applications in a daily fleet assignment*, Transportation Sci., 31 (1996), 237–248.

Our papers on VLSN search algorithms are available at the Web site www.ise.ufl.edu/VLSN.

Ravindra K. Ahuja is a professor of industrial and systems engineering at the University of Florida, Gainesville. James B. Orlin is a professor of operations research at the MIT Sloan School of Management and co-director of the MIT Operations Research Center.