

# A Bundle Algorithm Approach for the Aircraft Schedule Recovery Problem During Hub Closures

Benjamin G. Thengvall • Jonathan F. Bard • Gang Yu

*Graduate Program in Operations Research, Department of Mechanical Engineering,  
University of Texas, Austin, Texas 78712-1063*

*Graduate Program in Operations Research, Department of Mechanical Engineering,  
University of Texas, Austin, Texas 78712-1063*

*Department of Management Science and Information Systems, Graduate School of Business,  
University of Texas, Austin, Texas 78712-1175*

*ben@calebtech.com • jbard@mail.utexas.edu • yu@uts.cc.utexas.edu*

---

A bundle algorithm is presented to solve a multicommodity network model for determining a recovery *plan* for a single carrier with multiple fleets following a hub closure. The algorithm is shown to provide feasible near-optimal solutions much more quickly than can be obtained using a standard commercial mixed-integer programming code (CPLEX). In this application, a bundle method is used to solve a Lagrangian relaxation of the integer programming formulation. The full algorithm includes heuristic techniques for finding feasible solutions from the solutions to the relaxed problems. Extensive computations were performed using data from Continental Airlines. The results show that the proposed approach provides faster times to optimality in some cases and always obtains feasible, near-optimal solutions for larger problems much more quickly than can be found using CPLEX. In addition, while a standard commercial code will provide only one solution, this approach provides multiple high-quality solutions.

---

## Introduction

When faced with a lack of resources, airlines are often unable to fly their published schedule. This is frequently the result of equipment failure, air-traffic-control restrictions, inclement weather, or crew shortages. When problems like these arise, corrective action must be taken in real time in a manner that salvages revenue, placates customers, and returns the airline to the original schedule in a timely fashion. This general situation is called the *airline irregular operations control problem*, and involves not only aircraft routing, but also crew scheduling, passenger recovery, maintenance scheduling, baggage handling and gate

assignments. The more specific problem of rerouting aircraft is termed the *aircraft schedule recovery problem*.

In this paper, we address a subset of the more general problem; namely, determining the best response when a hub of an airline is closed. When an airport through which a large number of flights travels is closed for any length of time, major disruptions to an airline's published flight schedule are unavoidable. Intelligently rescheduling aircraft in such situations can save airlines thousands of dollars and minimize the adverse impact on passengers. Hub closures are most often the result of inclement weather and they represent one of the most extreme disruptions an airline may face. While they are not a frequent

occurrence—because of their tremendous disruptive impact on an airline’s operation—they must be considered carefully.

Teodorovic and Stojkovic (1990), Jarrah et al. (1993), and Argüello et al. (1997b) have approached the aircraft schedule recovery problem for a single fleet using network models. Another paper by Argüello et al. (1997a) presents a greedy randomized adaptive search procedure (GRASP) to find solutions. For almost all major airlines, however, the full problem involves multiple aircraft types. Carriers operate a variety of fleets and subfleets that have differing characteristics that limit their use. For example, some planes may be too large to service certain stations. Other planes may not be approved for flight over large bodies of water. Different types of aircraft also have different passenger carrying capacities. The general problem does not decompose into separate problems for each fleet and subfleet because substitution is allowed between some fleets and within fleets according to subfleet hierarchies. To address the more relevant multifleet case, Yan and Tu (1997) and Thengvall et al. (2001) have used multicommodity network flow models. In these models, each aircraft type is viewed as a separate commodity in the network.

The purpose of this paper is to present a bundle algorithm to solve a multicommodity network model for determining a recovery schedule for all aircraft operated by a large carrier following a hub closure. A bundle algorithm is an extension of the traditional subgradient algorithm in which past information is used collectively to find the current search direction. After introducing the model in §1, the bundle approach is described in §2. Methods for finding feasible solutions are given in §3. These two components are combined in §4 to give the complete algorithm, as implemented, to solve the aircraft recovery problem. Using data provided by Continental Airlines, extensive computational results are reported for a set of test problems corresponding to three hub closures. The results are then compared with those obtained from CPLEX’s mixed-integer programming solver. Section 5 briefly discusses some implementation issues such as step size strategy and the inclusion of null steps, and gives some computational examples that support the choices made in the algorithmic design.

## 1. Model Description

The problem addressed in this paper is that of re-scheduling aircraft during a specified recovery period in response to the closing of a hub. From the time the station closes until it reopens, no transient activity is permitted. Thus, given the position of planes at the closure time, the original flight schedule, the time of station closure and reopening, and a time set for recovery, we are interested in finding the “best” assignment of flights to all available aircraft such that once operations resume, all flights can be flown as originally scheduled. While in most cases, the length of the hub closure is not known in advance, this type of model is still relevant. In practice, real-time decisions are required, so estimates are made and solutions are implemented. As time passes and information that is more accurate becomes available, models can be rerun to update solutions. An example of this type of rolling horizon approach can be found in Bard et al. (1998).

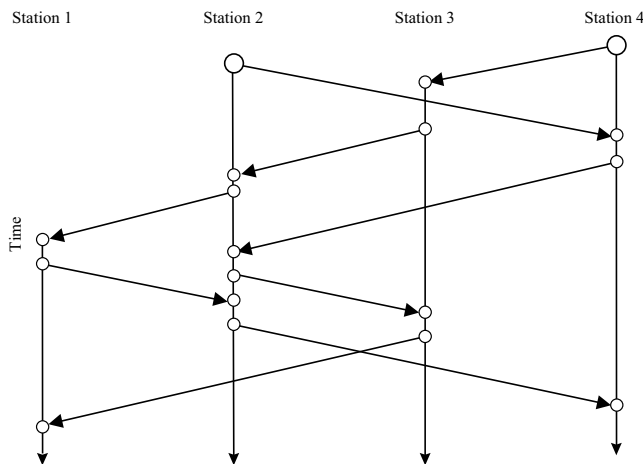
The model used for testing the bundle algorithm presented in this paper was first introduced by Thengvall et al. (2001). Previous testing relied on a standard commercial code (CPLEX) and so provides a benchmark by which to measure the benefits of the new approach. An abbreviated description of the model is given below. For a full description including solution characteristics, see Thengvall et al. (2000, 2001).

Many factors serve to complicate the problem of finding an optimal or best aircraft rerouting. One is the variety of aircraft operated by most major carriers. Another is the goal of recovering the schedule in a finite time period. Recovery by time  $t$  is defined as having the appropriate aircraft in place to fly all flights as scheduled from time  $t$  onward. Therefore, a recovery schedule must be built that ensures all aircraft will be at correct stations by certain times to achieve this goal. This constraint is called aircraft balance at the end of the recovery period.

Our model is derived from the work of Yan and Tu (1997) and solves the aircraft recovery problem for multiple fleets with the objective of maximizing a modified “profit” function over the flight schedule during the recovery period. The model allows flight

delays and cancellations, and accounts for passenger revenues in the objective function. In addition, it includes an incentive to minimize deviation from the original aircraft routings. The result is an integer multicommodity network model with side constraints which is NP-hard because of the addition of side constraints to the otherwise pure network structure (Garey and Johnson 1979).

The proposed model can be thought of as a collection of simple space-time networks that are connected with a set of flight cover constraints. Each aircraft equipment type (subfleet) has a single space-time network, as illustrated in Figure 1. In the diagram, the vertical axis represents time and the horizontal axis represents space. Flows on the network are aircraft. The vertical arcs represent aircraft on the ground that are waiting for a flight or that have no more flights during the recovery period. In the diagram, all flow is from top to bottom. Arrows are omitted on the ground arcs to avoid clutter. The sloping diagonal arcs represent flights from one station to another. Diagonal flight arcs are restricted to binary values while vertical ground arcs, which are only required to be nonnegative, will contain integer flow by construction. This can be seen from Figure 1. Beginning with integer values on the vertical arcs, because only binary flows can enter or leave each station via flight arcs, integrality will always be maintained on the vertical arcs. In some instances, not all flights will be covered. Flight arcs with no flow in the final solution are canceled.

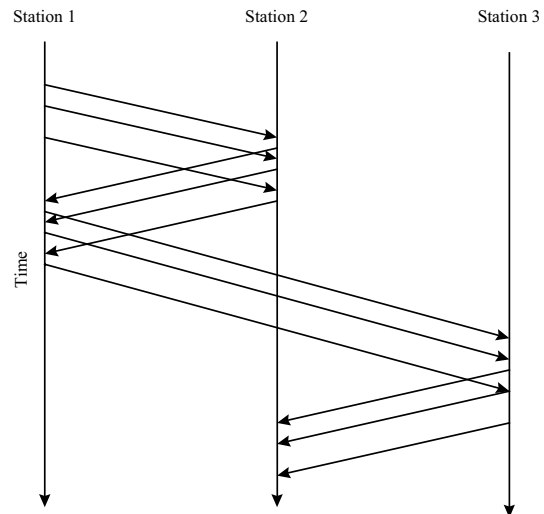


**Figure 1** Space-Time Network Representation

The two larger nodes in Figure 1 at Stations 2 and 4 are supply points for the model. Of course, each station need not have a supply. The model presented can accommodate recovery periods of arbitrary length. Aircraft are supplied to the model at various times depending on when they become available. If the recovery period begins in the middle of the day, as it often does, supply nodes will contain all aircraft on the ground at that time. Using intermediate supply nodes, planes in the air can be added to the model at the time they are scheduled to reach their current destination. The smaller nodes at each arc endpoint are the balance of flow (intermediate) nodes and are present at each flight departure or arrival.

**Incorporating Delays**

To account for delays on a particular flight leg, a series of flight arcs is created to represent the available options for taking the flight at a later time. In Figure 2, two delay options are shown for each of the four flight legs along with the originally scheduled on-time option. To account for the time it takes to prepare each plane for another flight, a turnaround time is added to each flight arc. To ensure that each leg is flown at most once, a cover constraint is added. This side constraint requires that the sum of flows on all arcs (variables) representing the same flight be less than or equal to one. The revenue for delayed flights



**Figure 2** Incorporating Delays

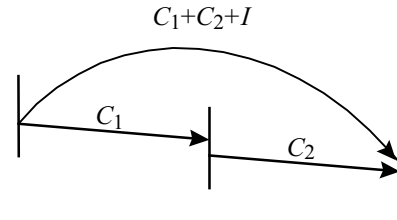
is adjusted to include a cost per minute delayed. Note that supply and intermediate nodes have been omitted in this and the following figures to reduce the amount of unnecessary detail.

### Extra Arcs for Discouraging Deviation in Aircraft Routings

In the airline industry, many issues are taken into account in the construction of the original flight schedule. Changes to aircraft routings affect crew schedules, gate assignments, scheduled maintenance, and passenger connections. Consequently, dealing with irregular operations in a way that encourages minimal deviation to the original aircraft routings is crucial. To “protect” original flight paths, additional flight arcs are incorporated in the general model. This new construct, represented by the circular arc in Figure 3a, will be called a *protection arc*. Protection arcs span a set of contiguous flight legs on an original flight path. The cover constraint introduced for delayed flights must now be expanded to include the extra flight option shown here. The new arc in Figure 3a corresponds to one plane flying both flight legs, rather than a single new flight. By providing an incentive on the new arc, a single plane is encouraged to fly both legs, thereby reducing the attractiveness of flying the two legs with separate aircraft. The use of a protection arc to discourage schedule deviation can be extended to a flight path containing any number of legs. As shown in Figure 3b, the protection arc added spans all three flight legs originally assigned to the aircraft. Here,  $C_3$  is the net revenue of Flight 3. The incentive on an arc that spans three flights should be greater than the incentive on an arc that spans two flight legs together. This is accomplished by adding a constant bonus value for each flight leg covered. Of course, each cover constraint, ensuring that each flight is assigned at most once, must be expanded to include all flight and protection arcs that represent that flight.

### Through-Flight Considerations

Through flights are mechanisms to serve long-haul markets with one or more intermediate stops, e.g., Boston-Newark-Los Angeles represents a through flight from Boston to Los Angeles. The revenues associated with



$C_1$ —passenger revenue minus cost for Flight 1.  
 $C_2$ —passenger revenue minus cost for Flight 2.  
 $I$ —incentive for plane assigned to Flight 1 to be assigned to Flight 2.

Figure 3a Protection Arcs

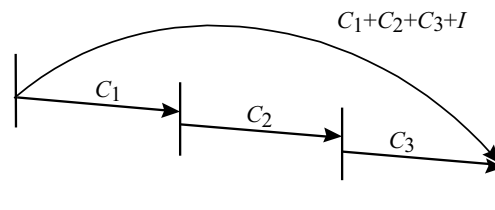


Figure 3b Minimizing Deviation from Original Schedule

flight legs that make up through flights are not independent. If one of the legs is canceled, the through-flight passengers will not make it to their destination as scheduled. To address this issue, an additional arc is added to the model that goes from the through flight’s origin to its destination. It does not represent a new flight option, but rather one aircraft assigned to all of the flight legs in the through flight. Again, this additional arc must be included in the cover constraint to ensure flights are not duplicated in the final solution. This construct can be applied to through flights with any number of intermediate stops.

Assume that the last two of the three flight legs shown in Figure 4 represent a through flight. Let  $0 < \pi < 1$  be the proportion of passengers that are taking both the second and third legs. If either leg is flown individually or if the legs are flown by different aircraft, the revenue gained will be discounted by the factor  $(1 - \pi)$ . If the new arc shown on the bottom is taken, the full revenue will be received for the through flight. Figure 4 includes the protection arcs from the previous section to demonstrate how the two constructs are used together and how the arcs are assigned revenue. Although many airline information systems track individual passenger connections

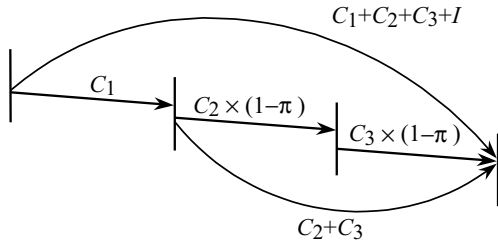


Figure 4 Inclusion of Through-Flight Arcs

in real time, including this data in an irregular operations model would make it too unwieldy to solve. However, when a large percentage of passengers are traveling the same set of flight legs, a through flight can be designated.

### Ferry Arcs

It is not always possible to recover the original schedule by a given time using only flights contained in an airline's original schedule. In such cases, ferry flights are necessary to achieve aircraft balance by the end of the recovery period. The model includes three types of ferry flights, as illustrated in Figure 5. Type I ferries

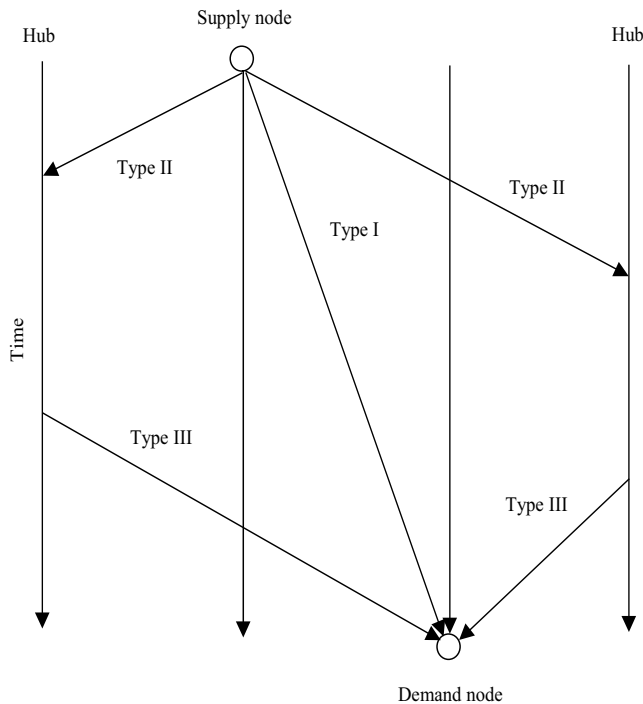


Figure 5 Type I, II, and III Ferries

connect each supply point with each demand point for a particular equipment type. Type II ferries go from each supply point to each hub. Type III ferries travel from each hub to each demand point.

### Considerations for Multiple Fleets

The network for each equipment type (subfleet) will contain all flights originally scheduled to be flown by that equipment type as well as delayed flights, protection arcs for flight paths, through flights, and ferry flights as described above. In addition, the network for each equipment type will also contain all flights, delayed flights, and through flights for which that equipment type can substitute according to the fleet substitution and subfleet hierarchy rules determined by the airline. Moreover, the cover constraints must be expanded to include these substitution arcs. Protection and ferry arcs appear only in the simple network associated with their original equipment types and are not involved in any substitutions.

A primary strength of this modeling approach is its ability to generate solutions that reflect changing user preferences. By adjusting the number of delay options, the cost per minute of delaying flights and the bonuses awarded for protecting flights, schedules with different average solution properties can be obtained. (Empirical results demonstrating this property for the case of a single fleet and minor disruptions are given by Thengvall et al. 2000.) Costs for interfleet and intrafleet substitution, as well as a cost for subfleet imbalance, can also be introduced.

### Mathematical Model

Sets.

- $E$  equipment types (subfleets)
- $F$  flight arcs (including delayed flights)
- $G$  ground arcs
- $I$  intermediate nodes (for flow balance and supply and demand)
- $P$  protection and through-flight arcs
- $R$  ferry arcs
- $N$  unique flight number representing each flight leg
- $O(\cdot, e)$  arcs originating at node  $(\cdot)$  for equipment type  $e$

$T(\cdot, e)$  arcs terminating at node  $(\cdot)$  for equipment type  $e$   
 $F(\eta)$  arcs covering flight  $\eta$ ;  $F(\eta) \subset F \cup P$

Parameters.

$C_{fe}$  passenger revenue minus flight cost minus delay cost minus substitution penalty for flight arc  $f$  flown by equipment type  $e$   
 $C_p$  for protection arc  $p$ —passenger revenues minus flight costs minus delay costs of all flights covered plus the appropriate incentive value; for through-flight arc  $p$ —passenger revenues minus flight costs minus delay costs of all flights covered  
 $C_r$  cost for ferry flight  $r$   
 $B_{ie}$  balance of aircraft at intermediate node  $i$  for equipment type  $e$  (integer valued if aircraft supplied or demanded at node  $i$ ; 0 otherwise)  
 $U_{ge}$  upper bound on ground arc  $g$  for equipment type  $e$

Variables.

$w_r$  binary flow on ferry arc  $r$   
 $x_{fe}$  binary flow on flight arc  $f$  for equipment type  $e$   
 $y_p$  binary flow on protection or through-flight arc  $p$   
 $z_{ge}$  flow on ground arc  $g$  for equipment type  $e$

$$\text{Maximize } \sum_{\substack{f \in F \\ e \in E}} C_{fe} x_{fe} + \sum_{p \in P} C_p y_p - \sum_{r \in R} C_r w_r, \quad (1a)$$

subject to

$$\begin{aligned} & \sum_{g \in G \cap O(i, e)} z_{ge} - \sum_{g \in G \cap T(i, e)} z_{ge} + \sum_{f \in F \cap O(i, e)} x_{fe} \\ & - \sum_{f \in F \cap T(i, e)} x_{fe} + \sum_{p \in P \cap O(i, e)} y_p - \sum_{p \in P \cap T(i, e)} y_p \\ & + \sum_{r \in R \cap O(i, e)} w_r - \sum_{r \in R \cap T(i, e)} w_r = B_{ie} \\ & \forall i \in I, \forall e \in E, \end{aligned} \quad (1b)$$

$$\sum_{\substack{f \in F \cap F(\eta) \\ e \in E}} x_{fe} + \sum_{p \in P \cap F(\eta)} y_p \leq 1 \quad \forall \eta \in N, \quad (1c)$$

$$w_r \in \{0, 1\} \quad \forall r \in R, \quad (1d)$$

$$x_{fe} \in \{0, 1\} \quad \forall f \in F, \forall e \in E, \quad (1e)$$

$$y_p \in \{0, 1\} \quad \forall p \in P, \quad (1f)$$

$$0 \leq z_{ge} \leq U_{ge} \quad \forall g \in G, \forall e \in E. \quad (1g)$$

The objective function (1a) maximizes the total profit associated with the recovery period schedule. The first term captures revenue, delay cost, and substitution cost on individual flight arcs. The second term duplicates these figures for protection and through-flight arcs. Recall that the protection arcs include an additional incentive value. The final term accounts for the cost of ferrying.

Flow balance at intermediate nodes on each sub-fleet network is maintained by constraint (1b). Equation (1c), the flight-cover constraint, assures that at most one arc associated with a particular flight receives flow. All ferry arcs (1d), flight arcs (1e), and protection arcs (1f) are binary variables. Ground arcs (1g) are required only to be nonnegative but will be integral in any feasible solution as previously explained. If a flight is canceled, no benefit is received or cost incurred. To account explicitly for cancellations, all that is necessary is to introduce a slack variable,  $s_\eta$ , in Equation (1c), a corresponding cost coefficient,  $C_\eta$ , and an additional term in the objective function of the form  $-\sum_\eta C_\eta s_\eta$ .

It is important to note that the objective function does not record true revenues and costs even if accurate figures are used for each flight leg. (This is one of the reasons why cancellations are not explicitly accounted for.) The addition of incentive values and delay costs makes the objective function artificial. Including these factors, though, allows us to weight the arcs so that the accompanying solution has favorable properties in terms of the number of cancellations, number of delays, and the amount of deviation to the original aircraft routings. More to the point, though, it is virtually impossible for airlines to provide accurate cost figures in real time. Hence, maximizing net revenue is problematic in any case. With regard to delay costs, the best that can be obtained are rough estimates without regard to such intangibles as passenger inconvenience and loss of customer goodwill. Cancellation costs are even more difficult to estimate, varying dynamically with passenger, crew and equipment disruptions.

## 2. Overview of Bundle Method

Bundle methods are a class of algorithms first introduced by Lemarechal and Mifflin (1978) to solve non-differentiable optimization problems. In our application, a bundle method is used to solve a Lagrangian relaxation of model (1) in which the flight-cover side constraints (1c) are moved to the objective function. Solutions to this relaxed problem represent an upper bound on the true objective value of the original maximization problem. Working with the relaxed problem, the issue is one of finding Lagrangian multipliers that will supply the tightest upper bound and of finding feasible solutions to the true problem given these relaxed solutions. Updating the multipliers to find the best upper bound is done using a bundle method approach. Obtaining feasible solutions from the relaxed problem will be covered in §3.

The original formulation is represented by the generic model in (2). The first set of equations represents the network flow constraints (1b). The second set of equations is the flight-cover side constraints (1c). These are the complicating constraints without which we would have a pure network flow problem. All variable bounds, Equations (1d)–(1g), still apply but are not shown here.

$$\begin{aligned} Z_{IP} = \text{maximize} \quad & cx, \\ \text{subject to} \quad & Ax = b, \\ & Dx \leq 1. \end{aligned} \tag{2}$$

The Lagrangian problem is given in (3), where  $u$  is a fixed, nonnegative row vector of conforming dimension. Solutions to this problem provide upper bounds on the true objective function. In addition, the transformation of the original problem results in a pure network that is much easier to solve.

$$\begin{aligned} Z_{LR}(u) = \text{maximize} \quad & cx + u(1 - Dx), \\ \text{subject to} \quad & Ax = b. \end{aligned} \tag{3}$$

To find the tightest upper bound, the following dual problem must be solved:

$$Z_{LD} = \underset{u \geq 0}{\text{minimize}} \quad Z_{LR}(u). \tag{4}$$

In the exposition, problem (3) will be treated as a black box, which, for any input  $\hat{u}$ , computes an

associated output  $Z(\hat{u})$  and one subgradient  $g(\hat{u})$ . In a bundle approach, the basic idea is to collect subgradient information at each step, storing it in the *bundle*,  $\beta$ . Using a combination of past subgradients, a new descent direction is determined at each step. The method described here follows the work of Lemarechal (1989) and Frangioni (1997).

In general, for any  $\hat{u}$ , there exists a subdifferential  $\partial Z(\hat{u})$  that is the set of all subgradients of  $Z(\hat{u})$ . The steepest descent direction can be found by solving

$$\min\{\|g\|^2: g \in \partial Z(\hat{u})\}. \tag{5}$$

If the subdifferential was known at every point, (5) could be solved and the steepest descent direction could be used at every step. The difficulty, however, is that the subdifferential is generally not available at any given point. All that is available is a single subgradient, which is not even guaranteed to be a direction of descent. In a bundle method, an attempt is made to estimate the subdifferential  $\partial Z(\hat{u})$  at any point  $\hat{u}$  using a combination of the subgradient at the current point and a collection of past subgradients.

At the current point, however, only the current subgradient is accurate. To deal with the inaccuracies of past subgradients approximate subgradients and a measure of their error must be defined. An  $\varepsilon$ -subgradient (of  $Z(\cdot)$  at  $\hat{u}$ ) is given by

$$Z(u) \geq Z(\hat{u}) + g(u - \hat{u}) - \varepsilon \quad \forall u \geq 0. \tag{6}$$

Using this idea, at any point  $\hat{u}$ , a measurement of the error associated with a subgradient from any past point  $u$  can be made by finding  $\alpha$ , the amount by which the subgradient inequality is violated:

$$\alpha = Z(u) - Z(\hat{u}) - g(u - \hat{u}). \tag{7}$$

At each step in a bundle algorithm, a convex combination of the subgradients in the bundle is used as the new search direction. Combining the ideas of (6) and (7) provides a measure of the quality of each subgradient in the bundle at the current point. The earliest bundle methods solved a problem like (8)

to determine a new search direction by providing weights  $\lambda_j$  for each  $g_j$  in the bundle.

$$\begin{aligned} \text{Min} \quad & \frac{1}{2} \left\| \sum_{j=1}^{|\beta|} g_j \lambda_j \right\|^2, \\ \text{subject to} \quad & \sum_{j=1}^{|\beta|} \alpha_j \lambda_j \leq \varepsilon \\ & \sum_{j=1}^{|\beta|} \lambda_j = 1, \\ & \lambda_j \geq 0 \quad \forall j. \end{aligned} \tag{8}$$

In this formulation, the  $\alpha_j$  can be viewed as weights on each subgradient which ensure that the less accurate the subgradient at the current point, the less influence it will have on the new search direction. The  $\varepsilon$  parameter places a limit on the total amount of inaccuracy allowed in the solution. To achieve good overall performance, dynamic updating of the  $\varepsilon$  parameter as the algorithm progresses is critical. Unfortunately, appropriate rules for this task have not been devised. Most recent approaches, including those proposed by Kiwiel (1990), relax the first constraint in (8) and solve the following quadratic program.

$$\begin{aligned} \text{Min} \quad & \frac{1}{2} \left\| \sum_{j=1}^{|\beta|} g_j \lambda_j \right\|^2 + \frac{1}{t} \sum_{j=1}^{|\beta|} \lambda_j \alpha_j, \\ \text{subject to} \quad & \sum_{j=1}^{|\beta|} \lambda_j = 1, \\ & \lambda_j \geq 0 \quad \forall j. \end{aligned} \tag{9}$$

As in problem (8), the convex combination of subgradients in (9) given by the weights  $\lambda_j$  provide the new search direction. The  $\varepsilon$  parameter has been removed in this simpler formulation and a multiplier  $1/t$  introduced. The interested reader can find a justification for using  $t$  as a step size as well as a number of rules for updating  $t$  in Frangioni (1999). A general scheme for the bundle algorithm is shown in Figure 6.

While a careful implementation of the above algorithm will converge to the dual objective  $Z_{LD}$  in (4), it will not necessarily provide feasible solutions to (2). Methods for obtaining feasible solutions are covered in §3. Values for  $u_0$  and  $t_0$  need to be defined as well

as a suitable strategy for updating  $t_k$  each iteration. In addition, it is necessary to specify a termination criterion. These issues are addressed in §4 where the complete algorithm, as implemented to solve (1), is given.

### 3. Finding Feasible Solutions

The side constraints (1c) that are moved to the objective contain the restriction that each flight may only be covered once. As the bundle algorithm progressively updates the vector of Lagrangian multipliers,  $u$ , the number of original flight-cover constraints violated (row violations) tends to decrease. Nevertheless, even as the bundle algorithm converges to the optimal objective value, the number of row violations rarely goes to zero. So while it will provide the optimal objective value to (4), it will not necessarily provide a feasible solution to the original problem. To find feasible solutions to problem (1), two additions have been made to the standard bundle algorithm. The first is a strategy for adding back-violated constraints to the relaxed problem (3). The second is a flow augmenting heuristic that can sometimes resolve violated constraints.

#### Adding Back Rows

One option for ensuring the enforcement of any given flight-cover constraint is to reintroduce it to the relaxed problem (3). When constraints are added to the problem, however, its size increases and it becomes more difficult to solve. If a small number of constraints is added, the problem can generally be reoptimized quickly using the dual simplex algorithm. A more serious issue is the loss of the pure network structure that introduces the possibility of nonintegral solutions. When nonintegral solutions result from the addition of constraints, a mixed-integer solver must be used to regain integrality. Solution times for mixed-integer problems can vary greatly.

Adding constraints is not a one-time fix for feasibility. Generally, when one set of constraints is added back to the model, another set of relaxed flight-cover constraints is violated. Therefore, adding back constraints is an iterative process. In our experience, as more constraints are added back, the number of violated rows in the solution to the relaxed problem

- Step 0.* Choose  $u_0$  and  $t_0$ . Solve (3). Set  $\beta = \{g_0\}$  and  $k = 1$ .
- Step 1.* Find search direction  $d_k = \sum_j \lambda_j g_j$  by solving (9).
- Step 2.* If movement along  $d_k$  yields enough improvement,  
     let  $u_k = u_{k-1} + t_k d_k$ . Update each  $\alpha$ . (Serious Step)  
     else  $u_k = u_{k-1}$ . (Null Step)
- Step 3.* Solve (6) and add  $g_k$  to  $\beta$ . If a null step is taken in Step 2, solve (6) as if a serious step were taken and add the  $g_k$  derived to  $\beta$ .
- Step 4.* Put  $k \leftarrow k+1$ . Update  $t_k$ , and repeat Step 1.

**Figure 6** Generic Bundle Algorithm

tends to decrease. On average, for the results shown in §4, 13% of the flight-cover constraints were added back to the relaxed problem (3) before optimality was achieved. (See Table 5 for the average number of flight-cover constraints for different problem sizes tested.)

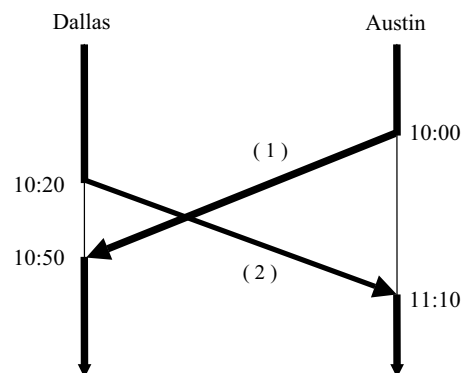
Once a flight-cover constraint is added back into the relaxed problem it remains in the relaxed problem. All subgradient values corresponding to constraints added back are fixed to zero for all subgradients in the bundle. The fact that these are no longer true  $\varepsilon$ -subgradients may, in general, hinder convergence to the true optimal Lagrangian multipliers; however, in practice these previously collected subgradients are useful in defining new search directions. A benefit worth noting of adding back rows is that the search space for the Lagrangian multipliers is reduced.

### Cross-Cancellation Heuristic

The cross-cancellation routine is a flow augmenting heuristic that attempts to resolve violated flight-cover constraints while maintaining balance of flow on the network. As explained in §1, the flight-cover constraints ensure that only one arc representing a given flight has positive flow in a solution. Therefore, when one of the flight-cover constraints (1c) is violated, there are two or more arcs representing the same flight with positive flow in the solution. To make such a solution feasible, flow must be restricted to only one of these arcs. By changing the flow on all but one offending arc from one to zero, we can satisfy the violated constraint. However, in doing so, the balance of flow in the network is disturbed. The cross-

cancellation heuristic determines whether the balance of flow can be restored by further augmentation of the flow on the network.

Once a violated constraint is identified, arcs with positive flow on the same subnetwork are examined one at a time to see if cross cancellation is possible. Figures 7 and 8 will be used to illustrate proper and improper cross-cancellation. The bold lines in the figures represent the flow on the network from the relaxed solution prior to implementing the heuristic. Let Arc 1 be an arc that could be canceled to satisfy the violated constraint. The origin (Austin) and destination (Dallas) stations along with the departure (10:00) and arrival (10:50) times are determined for this arc. Then a search is made for a single arc to cancel in conjunction with Arc 1. Let us consider Arc 2. A sufficient condition for canceling both arcs and maintaining balance of flow is that this second arc *cross* the first. That is, it has opposite origin and desti-



**Figure 7** Proper Single Cross-Cancellation

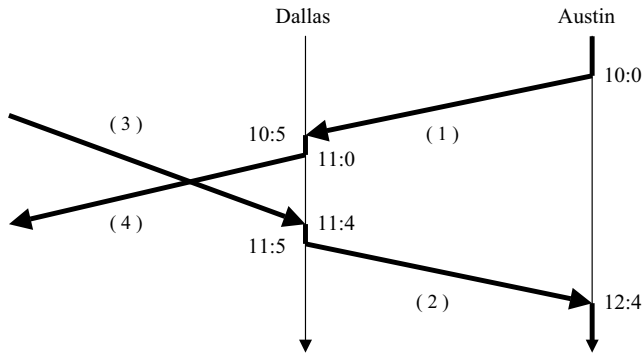


Figure 8 Improper Single Cross-Cancellation

nation stations and it departs before the first arc's arrival and arrives after the first arc's departure. This check ensures that if cross-cancellation occurs, flow that would have traveled on Arc 2 will be present to pick up flow that would have come from Arc 1 and vice versa. If such an arc can be found, canceling both arcs will result in balance of flow and the flight-cover side constraint that was violated will be satisfied. Figure 7 shows an instance where Arcs 1 and 2 can be canceled and balance maintained. Figure 8 provides an example where Arcs 1 and 2 do not cross. If Arcs 1 and 2 are canceled, balance of flow is maintained in Austin, but not in Dallas. If Arc 1 is canceled, Arc 4 will not have the flow that it needs. Flow will arrive on Arc 3 at 11:50, but it is too late to cover Arc 4 which departs at 10:50. The heuristic using only single cross-cancellation is outlined in Figure 9.

This cross-cancellation principle can be generalized to a set of any number of arcs subject to the following conditions:

- The first arc in the set begins at the destination station of the arc chosen for cancellation and departs prior to its arrival.
- Each subsequent arc in the set crosses the preceding arc.
- The final arc in the set terminates at the origin station of the arc chosen for cancellation and arrives after its departure.

By canceling such a set of arcs as well as the arc that violates the flight-cover constraint, balance of flow is maintained and the violated constraint is satisfied. Figure 10 shows an example of proper cross-cancellation where Arcs 2 and 3 are matched against the arc that could be canceled to satisfy the violated constraint (Arc 1). If all three arcs are canceled, balance of flow will be maintained. To include two or more cross-cancellations, Step 3 of the above algorithm would be modified as follows. Before declaring the algorithm unsuccessful, check for the possibility of two cross-cancellations, three cross-cancellations, and so on.

For the implementation described in this paper, up to four cross-cancellations were used. Cross-canceling more arcs than this was found to be counterproductive for two reasons. First, with five or more cross-cancellations the time taken for the heuristic to search through all possible combinations of arcs to cancel

*Input:* Solution to (3) that includes violated flight cover constraints.

*Output:* If successful, modified solution with violations resolved, else indication that heuristic failed.

- Step 1.* Identify an arc (Arc 1) representing a double-covered flight and record its origin, destination, departure, arrival, and subfleet.
- Step 2.* From the set of all arcs with unit flow in the same subfleet, search for an arc (Arc 2) that crosses Arc 1.
- Step 3.* If such an arc is found, change the flow on both Arc 1 and Arc 2 to zero if more violated flight cover constraints exist, return to Step 1, else heuristic is successful.  
 else, heuristic is unsuccessful.

Figure 9 Single Cross-Cancellation Heuristic

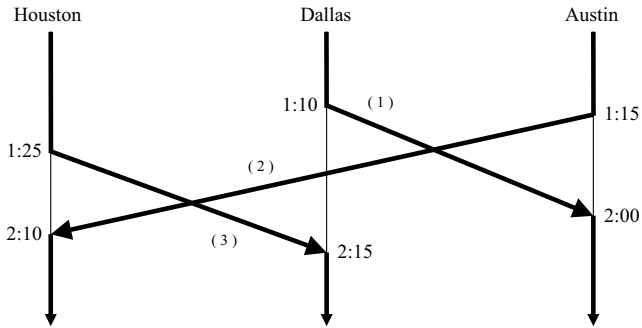


Figure 10 Proper Double Cross-Cancellation

becomes significant. Second, the more arcs that are canceled to obtain feasibility, the lower the objective function value. To obtain a feasible solution, the cross-cancellation heuristic must be successful for every violated row in the relaxed solution. Cross-cancellation is not always possible.

It should be noted that in its effort to maintain balance of flow the cross-cancellation heuristic checks for a sufficient condition, not a necessary one. Checks that are more complex could be implemented and a more complex heuristic could be developed to identify more solutions that are feasible. For example, in Figure 8 if there was no flow on Arcs 3 and 4, Arcs 1 and 2 could be canceled and balance of

flow maintained. Introducing such additional checks would increase the time complexity of the heuristic.

#### 4. Complete Algorithm and Computational Results

Combining the bundle algorithm presented with the strategies for obtaining feasible solutions, it is possible to describe the complete solution process. Convergence is measured by the size of the optimality gap which is calculated as (best upper bound – best feasible solution)/best upper bound, i.e.,  $(Z_{LR}(u) - Z_{feasible})/Z_{LR}(u)$ . This is essentially the same way the CPLEX MIP solver calculates the optimality gap. The complete algorithm is shown in Figure 11. The following three parameters are used in the algorithm:  $\tau$  is the threshold below which the feasible solution-finding subloop (Steps 4 and 5) is entered,  $\varphi$  is the number of times the subloop may be repeated without an iteration of the main loop, and  $\delta$  is used to control the rate of decrease of the step size  $t$ .

In Step 1, gradients that have not been used to form the new search direction,  $d$ , in more than 15 iterations are removed from the bundle. This keeps the size of the quadratic problem (9) from becoming too large. The search for feasible solutions is controlled by

- Step 0. Let  $u_0 = 0$ ,  $t_1 = 600$ ,  $Z_{LR}(u) = \infty$ , and  $Z_{feasible} = 0$ . Solve (3). Set  $\beta = \{g_0\}$  and  $k = 1$ .
- Step 1. Find search direction  $d_k = \sum_j \lambda_j g_j$  by solving (9). Eliminate unused gradients from  $\beta$ .
- Step 2. Let  $u_k = u_{k-1} + t_k d_k$ . Solve (3), and add  $g_k$  to  $\beta$ . Update  $Z_{LR}(u) = \min \{Z_{LR}(u), Z_{LR}(u_k)\}$ . If row violations = 0, update  $Z_{feasible} = \max \{Z_{feasible}, Z_{LR}(u_k)\}$  and go to Step 6.
- Step 3. If row violations  $< \tau$ , set rowadds = 1 and go to Step 4, else go to Step 6.
- Step 4. Add violated rows to (3) and solve. Attempt cross-cancellation heuristic. If successful and objective larger than old  $Z_{feasible}$ , update  $Z_{feasible}$ .
- Step 5. If rows still violated and rowadds  $< \varphi$ , put rowadds  $\leftarrow$  rowadds + 1 and go to Step 4, else go to Step 6.
- Step 6. If optimality gap  $> 0.01$ , put  $k \leftarrow k + 1$ ,  $t_k \leftarrow t_0 - (t_0 / \delta) \times k$ , and go to Step 1; else stop.

Figure 11 Full Bundle Algorithm

the number of violated rows in the current solution. Once this number drops below  $\tau$ , Steps 4 and 5 of the algorithm are executed where rows are added back and a search is made for feasible solutions using the cross-cancellation heuristic. For the larger test cases,  $\tau$  must be set higher to obtain convergence. The value of  $\tau$  should be set in practice at the point where little improvement is seen in the number of violated rows for a number of iterations of the bundle algorithm.

Parameter values that vary for different problem instances are shown in Table 1. Information on average problem sizes is found in Table 5. The parameter  $\varphi$  is set to 12 for all problem instances. A golden search was implemented to find an initial value for the step size  $t$ . A starting value of  $t_1 = 600$  was found to be reasonable for all problem instances. Step 6 shows the step size update formula. This formula provides a linearly decreasing step size. The actual number of iterations the algorithm goes through varies. If the algorithm continues for more than  $\delta$  iterations, the step size is set to a randomly generated number between 5 and 16. This is done to ensure the step size remains positive and is sufficiently small. However, more than  $\delta$  iterations are rare. For the larger test cases, a larger  $\delta$  value is used to provide a more gradual decrease in step size (see Tables 1 and 5). If  $\delta$  is too small, the step size decreases too rapidly and the algorithm may not converge.

CPLEX 6.0 is used to solve all optimization problems that arise in the implementation of the bundle algorithm. Problem (9) is a quadratic program and is solved using the barrier code. Problem (3) is solved

initially using the network code. After a few iterations, we switch over to the primal simplex algorithm. Once rows are added back into the network model, problem (3) is solved using the dual simplex algorithm. With the addition of rows, the pure network structure is lost and noninteger solutions are possible. When a solution returned by the dual simplex algorithm contains noninteger values, the mixed-integer solver is called.

The network optimizer available with CPLEX 6.0 does not have the ability to start from an advanced basis. That is, each time the network algorithm is used, problem (3) is resolved from scratch. The primal and dual simplex codes, though, are able to start from an advanced basis. The procedure described above that switches from the network algorithm to the primal simplex algorithm to the dual simplex algorithm provides the best solution times at the different phases of the overall bundle algorithm. It is believed that a network algorithm that was able to utilize an advanced basis would improve the overall performance time of the bundle algorithm. Such a network algorithm would be used for the solution of problem (3) until the first time violated rows were added back to the problem. At that point, the dual simplex algorithm would take over the computations.

The data used for testing were provided by Continental Airlines, which operates three hubs: Houston, Newark, and Cleveland. For the given schedule, 49% of flights have Houston as their origin or destination, 35% have Newark as their origin or destination and 14% have Cleveland as their origin or destination. The data spans  $2\frac{1}{2}$  days and includes 332 active aircraft from 12 different fleets. Each fleet has from one to six subfleets for a total of 28 different types of aircraft. The schedule includes 2921 flights between 149 domestic and international destinations.

To test the efficiency of the bundle algorithm a set of test cases was developed. The set consists of closing either the Houston, Newark, or Cleveland hub beginning at 10 a.m. for a fixed number of minutes, denoted by DOWNTIME, with a recovery period of RECOVER minutes after closing. As outlined below, nine scenarios were considered for each hub. Table 1 illustrates these nine instances. The first instance, for example, represents a downtime of 120 minutes and

**Table 1** Input Parameters

Instance	Row Violation Threshold $\tau$	Step Size Control $\delta$
120-240	15	15
120-360	15	40
360-480	20	50
360-660	20	50
360-840	20	50
600-720	20	60
600-960	20	60
600-1,200	20	60
600-1,400	25	60

a recovery period of 240 minutes. The hub is closed between 10 and 12 a.m. No flights can enter or leave during this time and all operations must be back on schedule by 2 p.m.

DOWNTIME = 120, 360, 600 minutes  
(2, 6, 10 hours);

RECOVER[120] = 240, 360 minutes  
(4, 6 hours);

RECOVER[360] = 480, 660, 840 minutes  
(8, 11, 14 hours);

RECOVER[600] = 720, 960, 1,200, 1,440 minutes  
(12, 16, 20, 24 hours).

These scenarios provide 27 total instances, 9 each at Houston, Newark, and Cleveland, on which to compare the performance of the proposed solution approach.

For comparison, each problem instance was solved with the bundle method and CPLEX's MIP solver applied directly to problem (1). For the CPLEX MIP implementation, the barrier algorithm was used to solve the initial (LP) relaxation, and the dual simplex algorithm was used during branch and bound. These options were chosen after much testing because they provided the fastest solution times. The stopping criterion for the search tree was set to 1% of optimality. That is, solutions are guaranteed to be within 1% of the true optimum. This same stopping criterion was used for the bundle method. Tables 2, 3, and 4 compare solution times for the two approaches for closures at each of the three hubs. All models were coded in C++ and solved on a Pentium 233 with 128 megabytes of RAM.

**Table 2 Solution Times (sec) for Houston Hub Closure**

Instance	CPLEX (MIP)	Bundle Algorithm	Bundle Feasible (% Optimality Gap)	Number Feasible	Average % Optimality Gap
120-240	22	26	25 (1.9)	2	1.3
120-360	68	80	72 (2.5)	2	1.8
360-480	234	201	126 (3.9)	2	2.6
360-660	493	440	291 (2.8)	9	1.9
360-840	539	358	237 (4.5)	12	2.5
600-720	411	657	247 (3.9)	19	2.8
600-960	472	442	259 (4.1)	11	3.0
600-1,200	459	879	291 (5.5)	50	2.4
600-1,400	1,241	1,144	748 (4.2)	16	3.1

**Table 3 Solution Times (sec) for Newark Hub Closure**

Instance	CPLEX (MIP)	Bundle Algorithm	Bundle Feasible (% Optimality Gap)	Number Feasible	Average % Optimality Gap
120-240	20	22	22 (4.0)	2	2.3
120-360	63	79	79 (0.1)	1	0.1
360-480	287	183	139 (2.4)	8	2.1
360-660	554	392	226 (2.8)	13	2.1
360-840	606	371	235 (2.3)	8	1.7
600-720	719	1,249	252 (4.3)	11	2.2
600-960	613	938	286 (4.5)	9	2.2
600-1,200	703	941	554 (2.4)	7	1.9
600-1,400	1,432	847	847 (0.4)	1	0.4

The columns labeled "bundle algorithm" show the time for the algorithm to converge to the 1% optimality gap. In 16 of the 27 cases (59%), the bundle algorithm converged faster than CPLEX's MIP solver. In instances where the hub was closed longer than 2 hours, 16 of the 21 cases (76%) converged faster with the bundle algorithm. The larger instances with the 360- and 600-minute closures represent the more difficult problems (see Table 5).

The columns labeled "bundle feasible (% optimality gap)" show the time to the first feasible solution and the percent optimality gap at that time. The "CPLEX (MIP)" columns show the time to converge to a 1% optimality gap, but this time is also the time to first feasible solution. Using CPLEX's standard mixed-integer algorithm on these problem instances, the time to optimality and the time to first feasible solution are identical. While this is not true in general, for the proposed model, the solution at the root node is often integral, and in cases where the root node

**Table 4 Solution Times (sec) for Cleveland Hub Closure**

Instance	CPLEX (MIP)	Bundle Algorithm	Bundle Feasible (% Optimality Gap)	Number Feasible	Average % Optimality Gap
120-240	21	25	23 (2.5)	2	1.5
120-360	58	72	46 (5.4)	3	2.3
360-480	289	157	139 (2.8)	5	1.8
360-660	573	469	310 (4.3)	6	2.6
360-840	666	320	316 (1.2)	2	0.6
600-720	560	353	315 (3.9)	8	2.0
600-960	568	400	264 (6.3)	9	2.6
600-1,200	650	408	385 (5.7)	8	2.7
600-1,400	2,162	984	748 (5.0)	7	4.2

**Table 5** Average Problem Information

Instance	Rows	Columns	Matrix Density	Side Constraints
120-minute closures	18,632	35,895	0.000108	430
360-minute closures	40,678	78,331	0.000048	867
600-minute closures	50,778	94,554	0.000038	1,098

is not integral it is rare that more than five or six nodes are searched before a solution is found. In all instances the first solution found was within the 1% optimality gap set for termination. CPLEX branch and bound heuristic options and cutting options were not employed. They were tested but generally increased the time to find solutions.

In every instance with a 360- and 600-minute closure, a feasible solution was found much more quickly using the bundle algorithm than using CPLEX's standard mixed-integer solver. On average, in the case of 360- and 600-minute closures, the time to a feasible solution with the bundle method was 334 seconds (over  $5\frac{1}{2}$  minutes) faster than the time to a feasible solution using CPLEX. As the aircraft recovery problem requires real-time solutions, the ability to obtain near-optimal, feasible solutions in less time is significant.

Another advantage of the bundle method is that it generates multiple, near-optimal feasible solutions. The columns labeled "number feasible" record the number of feasible solutions found by the bundle algorithm including the optimum. The columns "average % optimality gap" show the average percent optimality gap of those feasible solutions. In almost all instances a number of high-quality solutions are available. Of the feasible solutions found, 49.6% were uncovered with the cross-cancellation heuristic. In a real-time operational environment, decision makers prefer to have several good alternatives from which to choose rather than a single option.

Table 6 shows summary statistics relating to the number of iterations in the bundle algorithm and the fraction of the total time spent in each phase of the bundle algorithm. Results are averages for the scenarios at each hub. The time spent in the cross-cancellation heuristic is not shown because it is only a fraction of a percent of the total time spent in the bundle algorithm.

**Table 6** Summary Implementation Statistics

Hub	Iterations	Percent of Total Time Solving (9)	Percent of Total Time Solving (3)	Percent of Total Time Solving (3) After Adding Back Rows
Cleveland	39	0.9	82.1	17.0
Newark	42	0.4	56.2	43.4
Houston	50	0.7	60.3	39.0

## 5. Implementation Issues and Choosing Parameter Values

The bundle-type algorithm described above differs from a standard implementation in two main ways. In our case:

- All steps are taken, not just improving ones; i.e., there are no null steps.
- Convergence is measured using the known optimality gap and does not rely on the magnitude of the direction vector.

In a standard bundle implementation, at each iteration a search direction combined with a step size produces a new trial point, i.e., a new set of Lagrange multipliers,  $u_k$  (see Step 2 of the Full Bundle Algorithm in Figure 11). If this point provides a large enough decrease in the objective function, the new subgradient obtained from this point is added to the bundle and the search continues at the next iteration from this new point. If the decrease is not large enough, the new subgradient obtained from this point is still added to the bundle but at the next iteration, the search continues from the previous point. When this happens it is called a null step. For our application, the use of null steps was found less efficient than taking all steps (see below) and hence, all steps are taken even those that lead to an increase in the objective function in (4).

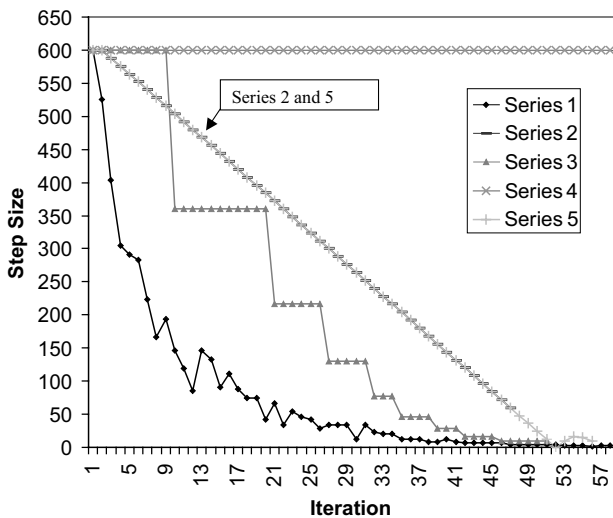
The use of the magnitude of the direction vector for convergence is more widely applicable because it does not require finding feasible solutions. However, for the aircraft recovery problem, the value of the objective function by itself is of little significance. If possible, it is better to measure convergence by comparing the best feasible solution found with the best upper bound, rather than only measuring convergence of the dual objective value in (3) as is normally done. We are able to take the former approach with the help of the strategies for finding feasible solutions given in §3.

Many implementation issues affect the speed and convergence of a bundle method. Figures 12–14 compare changes in step size  $t_k$ , objective value  $Z_{LR}(u_k)$ , and number of row violations using five different parameter settings. These settings (Series 1–5 below) represent different combinations of step size strategy, and taking all steps vs. using null steps. Although these examples are not exhaustive, and many other step-size selection rules from the literature and of our own invention were explored, they amply illustrate the range of possibilities. All results shown are for the 360–660 Houston hub closure. Similar results hold for the other test instances. A time limit of 2,500 seconds was placed on the computations. Further discussion regarding different bundle implementations can be found in Frangioni (1997, 1999).

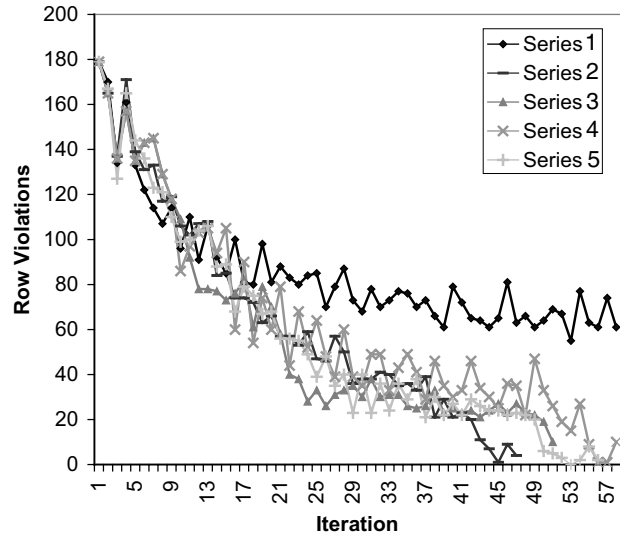
*Parameter Settings.*

- Series 1: Golden search (line search at each iteration)/All steps taken
- Series 2: Steady decrease (as described in §4)/All steps taken
- Series 3: If no improvement, decrease step size 40%/Null steps taken
- Series 4: Fixed-step size (step size = 600)/Null steps taken
- Series 5: Steady decrease (as described in §4)/Null steps taken

In the computations, the number of row violations gives a measure of the distance from a feasible solu-



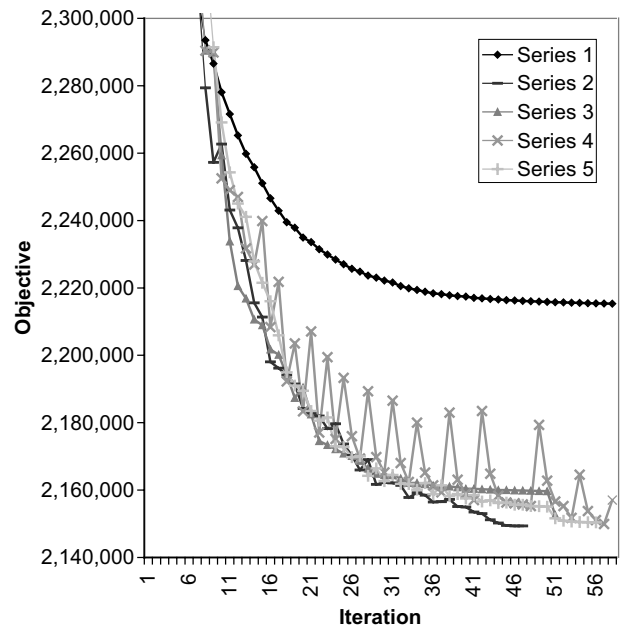
**Figure 12** Step Size per Iteration



**Figure 13** Row Violations per Iteration

tion. Figure 13 shows the number of row violations at each iteration for the five parameter settings, while Figure 14 shows the convergence of the bundle algorithm to the true objective value.

Examining Figures 12–14 and Table 7, it can be seen that Series 2 outperforms the others. Looking at Table 7, Series 2 takes less total time to converge,



**Figure 14** Objective Value per Iteration

**Table 7** Comparison of Five Parameter Configurations

	Series 1*	Series 2	Series 3	Series 4	Series 5
Total time	2,500	475	698	911	1,305
Number of iterations	58	47	51	61	55
Number of rows added	0	123	204	178	178
Best feasible solution	—	2,128,780	2,137,630	2,130,180	2,135,290
Best upper bound	2,215,760	2,149,340	2,152,620	2,148,500	2,148,290
Optimality gap (%)	1	0.0096	0.0070	0.0085	0.0060
Time to feasible solution	—	297	519	361	572
First optimality gap (%)	—	0.038	0.034	0.050	0.030

Note. \*No convergence after 2,500 seconds.

requires fewer iterations, adds back fewer rows, and finds a feasible solution faster than any other series. Series 1 was halted prior to convergence after 2,500 seconds. Series 3, 4, and 5 reach optimality eventually, but take longer to do so than Series 2. Series 2 is the implementation used to obtain the results presented in §4.

## 6. Summary

A modified bundle algorithm is presented to solve a multicommodity network-type model for determining a recovery schedule for all aircraft operated by a large carrier following a hub closure. The full methodology includes heuristics for finding feasible solutions from the solutions to these relaxed problems. On average, for the larger test cases, the algorithm finds a feasible solution twice as fast as a standard commercial code. Obtaining a high-quality feasible solution in half the time is a significant improvement for this complex, real-time application. Another major contribution of the proposed approach is its ability to generate multiple near-optimal solutions. Often, a number of practical constraints cannot be embedded in the model because of their scenario and time-dependent nature. Having multiple solutions provides decision makers with needed flexibility during the aircraft schedule recovery process.

While testing has shown the bundle approach to be effective, it is likely that better results could be

achieved with an improved version of CPLEX's (or the adaptation of any other) pure network solver that accepted an advanced basis during reoptimization. In an operational environment, a trade-off always exists between the quality of a solution and the time it takes to find it. If optimization methods are put to use in such environments, solution times must be measured in minutes rather than hours.

## References

- Argüello, M. F., J. F. Bard, G. Yu. 1997a. A GRASP for aircraft routing in response to groundings and delays. *J. Combinatorial Optim.* 5 211–228.
- , ———, ———. 1997b. Models and methods for managing airline irregular operations aircraft routing. G. Yu, ed. *Operations Research in the Airline Industry*. Kluwer Academic Publishers, Boston, MA, 1–45.
- Bard, J. F., L. Huang, P. Jaillet, M. Dror. 1998. A decomposition approach to the inventory routing problem with satellite facilities. *Transportation Sci.* 32 189–203.
- Frangioni, A. 1997. Dual-ascent methods and multicommodity flow problems. Ph.D. dissertation, Dipartimento di Informatica, Università Degli Studi di Pisa, Italy.
- . 1999. A bundle dual-ascent approach to linear multicommodity min-cost flow problems. *INFORMS J. Comput.* 11 370–393.
- Garey, M. R., D. S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Company, San Francisco, CA.
- ILOG, Inc. 1997. *Using the CPLEX Callable Library*, Version 5. CPLEX Division, Incline Village, NV.
- Jarrah, A. I. Z., G. Yu, N. Krishnamurthy, A. Rakshit. 1993. A decision support framework for airline flight cancellations and delays. *Transportation Sci.* 27 266–280.
- Kiwiel, K. C. 1990. Proximity control in bundle methods for convex nondifferentiable minimization. *Math. Programming* 46 105–122.
- Lemarechal, C. 1989. Nondifferentiable optimization. G. L. Nemhauser, A. H. G. Rinnooy Kan, M. J. Todd, eds. *Handbooks in Operations Research and Management Science*, Vol. 1, Optimization. North-Holland, Amsterdam, 529–572.
- , R. Mifflin, eds. 1978. *Nonsmooth Optimization*. Pergamon Press, Oxford, U.K.
- Teodorovic, D., G. Stojkovic. 1990. Model for operational daily airline scheduling. *Transportation Planning and Technology* 14 273–285.
- Thengvall, B. G., J. F. Bard, G. Yu. 2000. Balancing user preferences for aircraft schedule recovery during airline irregular operations. *IIE Trans. Oper. Eng.* 32 181–193.
- , ———, ———. 2001. Multiple fleet aircraft schedule recovery following hub closure. *Transportation Res.* 35A 289–308.
- Yan, S., Y. Tu. 1997. Multifleet routing and multistop flight scheduling for schedule perturbation. *Eur. J. Oper. Res.* 103 155–169.

Received: May 1999; revision received: May 2001; accepted: February 2002.