Dynamic Network Flow with Uncertain Arc Capacities: Decomposition Algorithm and Computational Results*

Gregory D. Glockner
ILOG, Inc.
Mountain View, California 94043

George L. Nemhauser Craig A. Tovey
School of Industrial and Systems Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332

May, 1997
Revised September, 1998

Abstract

In a multiperiod dynamic network flow problem, we model uncertain arc capacities using scenario aggregation. This model is so large that it may be difficult to obtain optimal integer or even continuous solutions. We develop a Lagrangian decomposition method based on the structure recently introduced in [9]. Our algorithm produces a near-optimal primal integral solution and an optimum solution to the Lagrangian dual. The dual is initialized using marginal values from a primal heuristic. Then, primal and dual solutions are improved in alternation. The algorithm greatly reduces computation time and memory use for real-world instances derived from an air traffic control model.

1 Introduction

Various real-world problems contain an underlying inventory structure with uncertain demand, capacity, costs, or availability. Although a decisionmaker needs to hedge against the unknown, computational limitations often make it difficult to incorporate uncertainty into a model. We demonstrate the viability of a scenario aggregation formulation [22] of a multistage network flow problem with uncertain arc capacities. This may be used to incorporate uncertainty in dynamic

*Supported by the Federal Aviation Administration under Research Grant 96-G-024

network flow models such as inventory models (§19.6 of [1]), air traffic management[7, 20, 25, 26], investment planning[16], railroad car distribution[12], and truckload scheduling[19].

With a scenario aggregation formulation, a real-world instance may be so large that it is difficult to obtain optimal integer or even continuous solutions. We develop a decomposition method that finds an optimum solution to a Lagrangian dual. In practice, our decomposition method also finds a nearly optimal primal integer solution. For real-world test problems derived from an air traffic control application [7], this method greatly reduces both the CPU time and the memory requirements when compared with linear solutions from commercial LP software. In the algorithm, marginal values from a primal heuristic give an initial dual solution. Then, the primal and dual solutions are updated alternately. Better dual solutions improve the primal solution, and better primal bounds improve the step size used by the dual optimization.

Following [9], a directed graph $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ is said to be dynamic if each node $i \in \mathcal{N}$ has a time $t(i)$ and each arc $a = (i, j) \in \mathcal{A}$ has $t(i) < t(j)$. A dynamic network flow problem is a network flow problem where the associated directed graph is dynamic. Denote the source-sink flow commodities using the index set $\mathcal{H} = \{1, \ldots, H\}$. For each commodity $h \in \mathcal{H}$, let b^h be the vector of surplus or deficit flow at the nodes, and let c^h be the vector of arc costs.

In our graph, we use random arc capacities to model uncertain real-world events. For example, consider a production system where the yield is uncertain due to equipment malfunction, variability in raw materials, or human error. We represent unfinished and finished goods using two sets of nodes. To represent a processing step, we use an arc with uncertain capacity that connects flow from an unfinished node to a finished node. More generally, random arc capacities can model uncertainty in many inventory systems, and according to §4.1 of [10], "inventory structures lie at the heart of a wide variety of dynamic [network] models. In an abstract sense, anything that involves some form of continued existence from one period to another may be conceived as an inventory flow."

More detailed examples are found in §1.2 of [8], and our dynamic network flow model for air traffic control is described in [7].

Correlation among random events is an essential characteristic of these models. For instance, flaws in raw materials may affect manufacturing yields for several weeks. Therefore, arc capacities can not be modeled as independent random variables. Instead, we model these uncertainties using a set of capacity scenarios, where each scenario represents a possible set of arc capacities in the graph. The set of scenarios branches from a common starting point, and subsets of scenarios branch from later points. For instance, a set of foggy weather scenarios might branch at noon, 1 p.m., and 2 p.m., representing potential times when fog might clear the airport. We use scenario aggregation[22, 29] to generate an immediate ("here-and-now") solution that hedges against multiple future scenarios. Using this scenario sample, we hope to generate a low-cost solution to the real-world system.

Specifically, let $\Omega = \{1, \ldots, K\}$ be an index set for the scenarios, and let p_k be the probability weight for scenario $k \in \Omega$. Let u^k be the arc capacities for scenario $k \in \Omega$, which is some realization of a random capacity vector $\mathbf{u}$. Assume that the problem is scaled so that $b^1, \ldots, b^H, u^1, \ldots, u^K$ are integers. Also, assume that the capacity on arc $a = (i, j)$ becomes known at time $t(i)$, the starting time for arc a. For convenience of notation, we refer to the start time of arc a by $t(a)$, where $t(a) = t(i)$. Let $\tau(k, k')$ be the latest time before scenarios k and k' branch. Specifically, if $\mathcal{A}_t = \{a \in \mathcal{A} : t(a) \leq t\}$ is the subset of arcs starting before time t, then

$$\tau(k, k') = \max\{t : u_a^k = u_a^{k'} \ \ \forall a \in \mathcal{A}_t\}.$$

Define x^{hk} as the variables for the flow of commodity h under scenario k. Let N be the node-arc

incidence matrix for $\mathcal{G}$. Our multistage stochastic program is

$$
\begin{aligned}
\min \quad & \sum_{h\in\mathcal{H}}\sum_{k\in\Omega} p_k(c^h\, x^{hk}) \\
& Nx^{hk} = b^h \qquad \forall h\in\mathcal{H};\forall k\in\Omega \qquad (1.1)\\
& \sum_{h\in\mathcal{H}} x^{hk} \le u^k \qquad \forall k\in\Omega \qquad (1.2) \qquad (1)\\
& x_a^{hk} - x_a^{hk'} = 0 \qquad \forall h\in\mathcal{H};\forall a\in\mathcal{A};\forall k,k'\in\Omega : t(a)\le\tau(k,k') \qquad (1.3)\\
& x^{hk} \ge 0 \qquad \forall h\in\mathcal{H};\forall k\in\Omega. \qquad (1.4)
\end{aligned}
$$

In (1), (1.1) are the flow balance constraints, (1.2) are the capacity constraints, (1.3) are the nonanticipativity constraints, and (1.4) are the nonnegativity constraints. The nonanticipativity constraints ensure that identical decisions must be made at any time when two scenarios are currently indistinguishable. Note that each scenario of (1) generates a multicommodity network flow problem. To obtain integer solutions, the nonnegativity constraint (1.4) is replaced by $x^{hk}\in Z_+^{|\mathcal{A}|}$. Even with one commodity, the integer flow problem is NP-complete (§5.2 of [8]).

This problem was introduced by Cheung and Powell, who call it a multistage stochastic program with network recourse. To solve (1), we may use linear programming decomposition techniques that exploit the block structure. There are several decomposition schemes that apply to general stochastic linear programs; see §III of [2] for an overview. Cheung and Powell consider decomposition techniques specifically for problems with network flow structure. They develop a solution strategy that decomposes the linear program (1) into time stages. This scheme, called tree decomposition[18], expresses a future value function in terms of trees in the network. They apply this method to a 2-period problem in [17] and to a multiperiod problem in [3]. Frantzeskakis and Powell develop a similar decomposition scheme in [6] that uses a linear approximation to the future value function.

Our modeling approach differs in two major ways from the methods of Cheung and Powell. First, Cheung and Powell study a flow model with a single commodity; we study a multicommodity flow problem. Second, the time-stage decomposition methods assume that the random arc capacities

are independent between time periods. In contrast, our scenario aggregation formulation (1) can have arbitrary correlations between time periods. The price for relaxing this assumption is that we must take care to select representative scenarios for (1).

In [9], we developed a new decomposition scheme called compath decomposition that dualizes the capacity constraints (1.2). This decomposition scheme is quite different from the progressive hedging method for general stochastic linear programs ([22], §6.3 of [2], and §2.9 of [13]), which dualizes the nonanticipativity constraints (1.3). In this paper, we develop a Lagrangian decomposition algorithm based on compath decomposition, and we show that this method is effective at solving (1).

We review compath decomposition in §2, which we use in a greedy primal heuristic in §3. In §4, we show how to obtain an initial dual solution using marginal values from our primal heuristic. In §5, we present nonsmooth optimization methods for optimizing the Lagrangian dual of (1). Finally, we describe the software implementation and present numerical results using data from an air traffic control model in §6.

2 Compath Decomposition

The various decomposition schemes for (1) are surveyed in [9]. These decomposition schemes are classified as temporal (time period) decomposition or constraint (Dantzig-Wolfe or Lagrangian) decomposition. We consider a constraint decomposition scheme where the capacity constraints (1.2) are dualized. This is similar to path decomposition for multicommodity flow problems, described in §17.4 of [1]. We introduced this scheme in [9] and called it *compath decomposition*.

Let π^k denote the dual variables associated with the capacity constraints (1.2) for scenario k.

Therefore,

$$
\begin{aligned}
L(\pi) \;=\; \sum_{k\in\Omega} \pi^k u^k + \;\min\; & \sum_{h\in\mathcal{H}}\sum_{k\in\Omega}(p_k c^h - \pi^k)x^{hk} \\
Nx^{hk} &= b^h \quad \forall h \in \mathcal{H}; \forall k \in \Omega \\
x_a^{hk} - x_a^{hk'} &= 0 \quad \forall h \in \mathcal{H}; \forall a \in \mathcal{A}; \forall k, k' \in \Omega : t(a) \le \tau(k,k') \\
x^{hk} &\ge 0 \quad \forall h \in \mathcal{H}; \forall k \in \Omega
\end{aligned}
\tag{2}
$$

is the Lagrangian resulting from relaxing the capacity constraints. Thus, (1) is equivalent to the Lagrangian dual problem

$$\max_{\pi\le 0} L(\pi). \tag{3}$$

To evaluate the Lagrangian (2), we solve one optimization subproblem

$$
\begin{aligned}
\min\quad & \sum_{k\in\Omega}(p_k c^h - \pi^k)x^k \\
Nx^k &= b^h \quad \forall k \in \Omega \\
x_a^k - x_a^{k'} &= 0 \quad \forall a \in \mathcal{A}; \forall k, k' \in \Omega : t(a) \le \tau(k,k') \\
x^k &\ge 0 \quad \forall k \in \Omega
\end{aligned}
\tag{4}
$$

for each commodity h. The subproblem (4) contains the flow constraints (1.1), nonanticipativity constraints (1.3), and nonnegativity constraints (1.4). The Lagrangian (2) is the compath master problem, and (4) is the compath subproblem.

Informally, a compath is a set of paths, one per scenario, such that the set satisfies the nonanticipativity constraints. We formalize this definition as follows. Let $\mathcal{A}_t = \{a \in \mathcal{A} : t(a) \le t\}$ be the subset of arcs that start before time t. We say that the source-sink paths q, q' are compatible for scenarios k, k' if $q \cap \mathcal{A}_{\tau(k,k')} = q' \cap \mathcal{A}_{\tau(k,k')}$. Thus, a set of paths $\{q^1, \ldots, q^K\}$ is a compath if each pair q^{k_1}, q^{k_2} is compatible for the corresponding scenarios k_1, k_2.

In [9], we prove the following structural results about the subproblem (4):

Compath Decomposition (Corollary 1 in [9]) *Let $\mathcal{C}(\Omega) = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_j\}$ be the set of compaths for the single-source, single-sink, dynamic, connected digraph $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ with capacity sce-*

narios Ω. Let $\mathcal{Y} = \{Y_1, Y_2, \ldots, Y_j\}$ be the set of source-sink flows corresponding to the compaths $\mathcal{C}(\Omega)$. Then $\mathcal{Y}$ is the set of all extreme points for the subproblem (4).

Cheapest Compath Algorithm *The subproblem (4) can be solved by assigning all flow to the cheapest compath, which can be obtained in time $O(K|\mathcal{A}|)$.*

The cheapest compath algorithm is based on a shortest path algorithm for acyclic graphs. Here, we view this subproblem algorithm as a "black box" as we study solution methods for the overall problem (1).

3 Primal Heuristic

¿From the compath decomposition corollary, any solution x to (1) can be decomposed by compaths. Reversing this process, we can build a solution by assigning flows along compaths. The primal heuristic greedily constructs a solution by augmenting the existing flow as much as possible along the cheapest feasible compath. The steps of this heuristic are summarized in Figure 1. In the heuristic, the cost vector $\tilde{c}^h$ determines the augmentation sequence. In step 1, we initialize with zero flow. Modifying the costs in step 9 ensures that step 5 generates a feasible compath. The cost function $c(\mathcal{C})$ in step 10 denotes the actual cost of compath $\mathcal{C}$. By design, the heuristic will generate an integral flow whenever the capacities u^k and total flow v_h are integral. With each augmentation, either the capacity becomes saturated for an arc in step 9 or v_h becomes zero. Thus, there can be at most $O(K|\mathcal{A}| + H)$ augmentations. Since we can find a cheapest compath in time $O(K|\mathcal{A}|)$, it follows that the heuristic takes $O(K^2|\mathcal{A}|^2 + HK|\mathcal{A}|)$ time. In practice, the number of commodities H is much smaller than $K|\mathcal{A}|$, so the performance is typically $O(K^2|\mathcal{A}|^2)$. Thus, the number of commodities generally has little effect on the running time of the heuristic.

1. $y_a^{hk} \leftarrow 0$ for all a, h, k
   Initialize cost vector $\tilde{c}^h$ for all h
   $v^h \leftarrow$ total source-sink flow for commodity h
   $n \leftarrow 0$, $z \leftarrow 0$
2. **for** $h \leftarrow 1$ to H
3. **while** $v_h > 0$
4. $n \leftarrow n + 1$
5. find cheapest compath $\mathcal{C}$ using costs $\tilde{c}^h$
6. $\delta \leftarrow \min \left\{ v_h, \min_{(a,k)\in\mathcal{C}} \left\{ u_a^k - \sum_{h\in\mathcal{H}} y_a^{hk} \right\} \right\}$
7. **for** $(a, k) \in \mathcal{C}$
8. $y_a^{hk} \leftarrow y_a^{hk} + \delta$
9. **if** $u_a^k = \sum_{h\in\mathcal{H}} y_a^{hk}$ **then** $\tilde{c}_a^{hk} \leftarrow \infty$ for all h
10. $z \leftarrow z + \delta\, c^h(\mathcal{C})$, $v_h \leftarrow v_h - \delta$

Figure 1: Primal Heuristic

The objective value, however, depends on the commodity order. The heuristic finds a solution with a good objective value when the commodities are pre-ordered by time span. Specifically, if s^h and t^h are the source and sink nodes for commodity h, then the commodities should be sorted so that $t(t^1) - t(s^1) \geq \ldots \geq t(t^H) - t(s^H)$. Ties are broken by sorting in increasing order of $t(s^h)$.

Even if (1) is feasible, the primal heuristic may not always find a feasible flow. However, the heuristic has never failed to find a primal feasible solution in our test problems. Moreover, we can ensure that the heuristic will find a feasible solution by introducing artificial arcs with high cost and infinite capacity. Often, these artificial arcs have a natural interpretation in the model. For instance, flow on these arcs can represent canceling a flight or shipping items directly from the factory to the customer. To determine if the primal is truly infeasible, we check for dual unboundedness as discussed in §5.

Initially, we run the heuristic with $\tilde{c}$ equaling the expected costs $(p_1 c, \ldots, p_K c)$. A set of primal

values are compared with LP optimal solutions in Table 1. In some cases, the difference between

Problem	LP optimum	Primal Values Obj	Diff	% Diff
atl50	23411.8	24764.6	1352.8	5.78%
atl100	23230.1	25064.4	1834.3	7.90%
atl150	25050.2	26781.3	1731.1	6.91%
atl250	26538.6	27810.6	1272.0	4.79%
dca50	63260.7	66350.6	3089.9	4.88%
dca100	57397.0	60982.2	3585.3	6.25%
dca150	61879.1	65728.9	3849.7	6.22%
den50*	15294.0	15417.7	123.7	0.81%
den100*	17316.9	17443.4	126.5	0.73%
den150	17406.0	17632.5	226.4	1.30%
den250	19009.7	19312.8	303.2	1.59%
mco50*	2419.4	2471.5	52.1	2.15%
mco100*	2605.7	2623.3	17.6	0.67%
mco150*	2751.5	2791.7	40.2	1.46%
mco250	3059.9	3099.0	39.0	1.28%
sea50	59847.5	65430.1	5582.6	9.33%
sea100	78537.9	85948.0	7410.1	9.44%
sea150	77971.5	84210.4	6239.0	8.00%
average				4.42%

Table 1: Initial Upper Bounds from Primal Heuristic

the LP and heuristic values is partly due to an integrality gap. Hopefully, the primal heuristic generates better values on problems with no integrality gap. This is confirmed by the test problems with no integrality gap, which we indicate by an asterisk in Table 1. For these problems, the average difference between the LP and heuristic values is only 1.16%. This suggests that the primal heuristic finds nearly optimum integral solutions.

Once dual information is available, we select a new cost vector $\tilde{c}$ and re-evaluate the heuristic to obtain a new primal solution. Following [28], we tried the cost vector $(p_1 c - \pi^1, \ldots, p_K c - \pi^K)$, where π^k are the dual variables associated with the capacity constraints for scenario k. This is similar to the cost vector used in the subproblem (4) resulting from evaluating the Lagrangian (2). Our heuristic yielded slightly better results when we used a line search to generate $\tilde{c}$ between the

initial vector $(p_1 c, \ldots, p_K c)$ and the vector $(p_1 c - \pi^1, \ldots, p_K c - \pi^K)$. Alternately, we could find a cost vector $\tilde{c}$ by performing a local search near the value $(p_1 c - \pi^1, \ldots, p_K c - \pi^K)$.

4 Initial Dual Solution

In our initial tests, we started the Lagrangian optimization from a naive solution like $\pi_a^k = 0$ or $\pi_a^k = -\epsilon$ for all k, a. The dual optimization converged very slowly because these values are far from optimum. Instead, we generate an initial solution via marginal values from the primal heuristic. This idea follows loosely from Wagelmans' survey [27] on sensitivity analysis in combinatorial optimization. To make the marginal value procedure efficient, we reorder the augmentations in the heuristic.

First, consider the economic interpretation of the dual variables π. At optimality, $\bar{\pi}_a^k$ represents the marginal value of the capacity constraint $\sum_{h \in \mathcal{H}} x_a^{hk} \leq u_a^k$. Thus, we can obtain an initial dual value π_a^k by determining the marginal value of the capacity constraint in the heuristic. Since the flows and capacities are integers, we determine the marginal value by perturbing the capacity value by one unit. Let $\tilde{c} = (p_1 c, \ldots, p_K c)$ be the initial cost vector in the heuristic, let R be the value of the primal heuristic, and let R_a^k be the value of the primal heuristic when the capacity u_a^k is replaced by $u_a^k - 1$. Thus, $R - R_a^k$ is the marginal value of the capacity constraint in the heuristic and $\pi_a^k \leftarrow \min\{0, R - R_a^k\}$ is an initial dual value. We call this the recomputation procedure since it recomputes the primal heuristic for each capacity perturbation. Since it takes $O(K^2|\mathcal{A}|^2)$ steps to compute the primal heuristic, it takes $O(K^3|\mathcal{A}|^3)$ steps to determine the initial dual solution via recomputation.

We can improve the running time by two simple speedup tricks. First, we can avoid computing values that we know will equal zero. Specifically, if a primal flow $\sum_{h \in \mathcal{H}} x_a^{hk}$ is strictly less than the

capacity u_a^k, then $\pi_a^k = 0$. Second, since the initial augmentations of R and R_a^k are identical, we can reduce redundant calculations in R_a^k by careful initialization.

We get an even faster procedure by approximating the marginal value calculation. Instead of augmenting flow by multiple units, think of the heuristic as augmenting one unit of flow at a time. For the original problem, suppose the heuristic augments sequentially along compaths $\mathcal{C}_{(1)}, \mathcal{C}_{(2)}, \ldots, \mathcal{C}_{(n)}$. Suppose the augmentation along compath $\mathcal{C}_{(m)}$ saturates arc a under scenario k. This implies that the compaths $\mathcal{C}_{(m+1)}, \mathcal{C}_{(m+2)}, \ldots, \mathcal{C}_{(n)}$ do not contain arc a under scenario k. Hence, the augmentation sequence $\mathcal{C}_{(1)}, \ldots, \mathcal{C}_{(m-1)}, \mathcal{C}_{(m+1)}, \ldots, \mathcal{C}_{(n)}$ is feasible for the perturbed problem. With this partial solution, we can augment along some compath $\mathcal{C}'$ to get a feasible flow in the perturbed problem. Essentially, this postpones the mth augmentation until the end of the heuristic. The difference between the original and perturbed solutions is the estimate of the marginal value. However, since the solutions only differ in compaths $\mathcal{C}_{(m)}$ and $\mathcal{C}'$, the net difference equals $c(\mathcal{C}_{(m)}) - c(\mathcal{C}')$. This approximation motivates the postponed recomputation procedure.

To determine the initial dual value π_a^k using the postponed recomputation procedure, start with the solution x that the primal heuristic generates for the original problem. Using a table, determine the augmenting compath $\mathcal{C}_{(m)}$ that violates the capacity on arc a under scenario k in the perturbed problem. Reduce the solution x by one unit of flow along the compath $\mathcal{C}_{(m)}$, then re-saturate arc a under scenario k to reflect the perturbation. Find $\mathcal{C}'$, the cheapest compath for the postponed augmentation. Therefore, $\pi_a^k \leftarrow \min\{0, c(\mathcal{C}_{(m)}) - c(\mathcal{C}')\}$ is the initial dual value. By storing the original solution and original compaths in a table, the major step is the computation of the new compath $\mathcal{C}'$, which takes $O(K|\mathcal{A}|)$ time. This means that postponed recomputation can compute all initial dual values in time $O(K^2|\mathcal{A}|^2)$.

The recomputation procedure usually gives a somewhat better initial solution than the postponed recomputation procedure. However, this difference quickly becomes insignificant once the

dual optimization procedure is underway.

5 Dual Optimization

The Lagrangian function (2) is continuous, concave, and piecewise linear, so we solve the Lagrangian dual (3) by nonsmooth optimization techniques. Since the standard nonsmooth optimization problem is convex, we solve the equivalent problem $\min\{\ell(\pi) : \pi \leq 0\}$, where $\ell(\cdot) \leftarrow -L(\cdot)$. Let

$$\begin{aligned} x^h(\pi) \;=\; \arg\min \sum_{k\in\Omega} (p_k c^h - \pi^k) x^k & \\ N x^k &= b^h \qquad \forall k \in \Omega \\ x^k_a - x^{k'}_a &= 0 \qquad \forall a \in \mathcal{A}; \forall k, k' \in \Omega : t(a) \leq \tau(k, k') \\ x^k &\geq 0 \qquad \forall k \in \Omega \end{aligned}$$

denote a subproblem solution obtained in evaluating the Lagrangian at π, and let

$$X(\pi) = \{(x^1(\pi), \ldots, x^H(\pi))\}.$$

For any π, $g(\pi) = -u + \sum_{h\in\mathcal{H}} x^h(\pi) \in \partial\ell(\pi)$ is a subgradient of $\ell(\pi)$. We may think of $g(\pi)$ as the amount that the subproblem solution violates the capacity constraints. Hence, the subdifferential of $\ell(\pi)$ is $\partial\ell(\pi) = \{-u + \sum_{h\in\mathcal{H}} x^h : (x^1, \ldots, x^H) \in X(\pi)\}$.

In each iteration of nonsmooth optimization, a solution π is updated to get $\pi' \leftarrow \pi + \alpha d$, where d is a direction and α is a scalar step size. First, we describe how we generate the search direction and step size. Then, we give methods to maintain dual feasibility so that $\pi' \leq 0$. Finally, we will discuss dual infeasibility and dual unboundedness.

The simplest search direction is the negative subgradient, i.e. $d = u - x(\pi)$. This technique, often called the subgradient method, is described in [14] and chapter 2 of [24] for general nonsmooth optimization problems. In our test problems, we found that the convergence was very slow. This seems to be caused by zig-zagging, a problem of cycling among several subgradients. Several other

direction methods try to reduce zig-zagging. Dilation methods ([14] and chapter 3 of [24]) project the subgradient using a deflection matrix. Bundle methods[14, 23] use an optimization subproblem to generate a search direction that is a convex combination of a finite set of previously generated negative subgradients. However, our large-dimensional direction problem may take an excessive amount of time to generate a search direction with dilation or bundle methods.

We use another search direction that is a convex combination of previously generated negative subgradients. Consider some collection $G = \{g^1, \ldots, g^r\} \subseteq \partial\ell(\pi)$ of subgradients of $\ell(\pi)$. We assume that $0 \notin \partial\ell(\pi)$ since otherwise π would be an optimum solution of $\ell(\cdot)$. Construct the search direction

$$d = -\frac{\sum_{g \in G} g/\|g\|^2}{\sum_{g \in G} 1/\|g\|^2}. \tag{5}$$

This direction is rarely the same as the direction generated using a bundle method. In our tests, however, (5) works well since it combines the behavior of the bundle method with the simplicity of the subgradient method.

In our implementation, we use a collection G of ϵ-subgradients of $\ell(\pi)$. The ϵ-subgradients are selected from the subgradients $g_{(1)}, g_{(2)}, \ldots, g_{(n)}$ found in previous iterations. Specifically, let $g_{(i)}$ be a subgradient of $\ell(\pi_{(i)})$. From the definition of a subgradient in chapter 23 of [21],

$$\ell(\lambda) \geq g_{(i)}(\lambda - \pi_{(i)}) + \ell(\pi_{(i)})$$

for all λ. If

$$\epsilon \geq g_{(i)}(\pi_{(i)} - \pi) - \ell(\pi_{(i)}) + \ell(\pi), \tag{6}$$

then it follows that

$$\ell(\lambda) \geq g_{(i)}(\lambda - \pi) + \ell(\pi) - \epsilon$$

for all λ. If $g_{(i)}$ is a subgradient of $\ell(\pi_{(i)})$ and (6) is satisfied, then $g_{(i)}$ is an ϵ-subgradient of $\ell(\pi)$. So we construct the collection G of ϵ-subgradients by taking the subgradients from previous iterations

that satisfy (6).

Unlike traditional nonlinear programming, the subgradient method does not generally incorporate a line search to determine an appropriate step size. However, the subgradient method will converge using one of the step size rules described in [14] and §4.3 of [15]. We use the step size rule developed in [11], which has a geometric convergence rate. Define the step size

$$\alpha \leftarrow \beta \frac{\bar{z} + \ell(\pi)}{\|d\|^2}, \tag{7}$$

where $\beta \in (0, 2)$ is a constant and $\bar{z}$ is the optimum value in the linear program (1) or the Lagrangian dual (3). Since $\bar{z}$ is unknown, we use the estimate $\hat{z} = (z_p + z_d)/2$, where z_p and z_d are the best current primal and dual values, respectively. Thus, the dual step size benefits from improved primal information. Although convergence is not guaranteed from the estimate $\hat{z}$, we always observed slower convergence with other step size rules that do not use an estimate of the optimum value $\bar{z}$.

Next, we must ensure that π' is dual feasible, i.e. $\pi' \leftarrow \pi + \alpha d \leq 0$. We assume that $\pi \leq 0$ so that we start from a feasible point. We tested penalty, barrier, and projection methods and found the projection methods to be the most effective in maintaining dual feasibility.

We use two basic projection methods. In the first, we project the direction d to create a feasible direction $\dot{d}$ where

$$\dot{d}_i = \begin{cases} 0 & \text{if } d_i > 0 \text{ and } \pi_i = 0 \\ d_i & \text{otherwise.} \end{cases} \tag{8}$$

In the second method, we project the iterate $\pi + \alpha d$ onto the negative orthant $\{\pi : \pi \leq 0\}$ by letting $\pi' \leftarrow [\pi + \alpha\, d]_-$, where

$$[(y_1, \ldots, y_n)]_- = (\min\{y_1, 0\}, \ldots, \min\{y_n, 0\}).$$

Used separately, each projection method has drawbacks. When projecting the direction, the step size α given in (7) may be infeasible. With a smaller, feasible step size, the algorithm may stall at

a suboptimal solution. This condition, known as jamming, occurs frequently in our test problems. When projecting the iterate, the step size rule (7) chooses a small step size because there are elements i with $d_i > 0$ and $\pi_i = 0$. However, combining these projections generally avoids these computational problems. First, we project the direction using (8) to get a feasible direction $\dot{d}$. Then we use $\dot{d}$ in (7) to get a step size. Finally, we project the iterate to get $\pi' = [\pi + \alpha\,\dot{d}]_-$.

Now, we consider unboundedness and infeasibility of the Lagrangian dual (3). The feasible region of the cheapest compath subproblem (4) used to evaluate the Lagrangian contains an acyclic shortest path problem for each commodity plus the nonanticipativity constraints, which are side constraints. If there exists a source-sink path, then the subproblem must have a finite solution; otherwise, the subproblem is infeasible. Thus, we obtain

Proposition 1 *The Lagrangian function (2) is defined for all π if and only if there exists a source-sink path for each commodity.*

¿From the proposition, the primal (1) and the Lagrangian dual are both infeasible if and only if there is no source-sink path for some commodity. For the Lagrangian dual to be unbounded, there must exist some π such that $L(\pi)$ is greater than a known upper bound $\check{z}$. We determine a value $\check{z}$ by considering solutions to the primal problem (1). The worst possible solution to the primal allocates all flow for each commodity to the most expensive compath. This compath is clearly bounded by the sum of the positive arc lengths. Thus,

$$\check{z} = \left(\sum_{h \in \mathcal{H}} v_h\right)\left(\sum_{a \in \mathcal{A}} [c_a]_+\right)$$

is an upper bound on the optimum solution, where v_h is the total source-sink flow for commodity h. If we want to use dual unboundedness as a test for primal infeasibility, then we cannot use the step size rule (7). In this case, we use one of the other step size rules described in [14] or §4.3 of [15].

1. $n \leftarrow 0$
2. $z_p \leftarrow R[(p_1 c, \ldots, p_K c)]$
3. $\hat{z} \leftarrow z_p$
4. find initial π by postponed recomputation
5. $z_d \leftarrow L(\pi)$
6. **while** $n < N$ and $z_p - z_d > \epsilon_1$
7. $\quad \pi_{(n)} \leftarrow \pi$
8. $\quad g_{(n)} \leftarrow u - \sum_h x^h(\pi)$
9. $\quad G \leftarrow \{g_{(i)} : L(\pi) - g_{(i)}(\pi - \pi_{(i)}) - L(\pi_{(i)}) \leq \epsilon_2\}$
10. $\quad d \leftarrow (\sum_{g \in G} g/\|g\|^2)/(\sum_{g \in G} 1/\|g\|^2)$
11. $\quad$ project d to get $\dot{d}$
12. $\quad \alpha \leftarrow \beta[\hat{z} - L(\pi)]/\|\dot{d}\|^2$
13. $\quad \pi \leftarrow [\pi + \alpha \dot{d}]_-$
14. $\quad z_d \leftarrow \max\{z_d, L(\pi)\}$
15. $\quad z_p \leftarrow \min\left\{z_p, \min_{0 \leq \alpha \leq 1}\{R[(p_1 c, \ldots, p_K c) - \alpha\pi]\}\right\}$
16. $\quad \hat{z} \leftarrow (z_p + z_d)/2$
17. $\quad n \leftarrow n + 1$

Figure 2: COMET (Compath Network) Algorithm

6 Computational Implementation and Results

We tested compath decomposition with data from the air traffic model described in [7]. This model contains a single sink with multiple sources, which allows us to construct both multicommodity and single commodity test problems. We used three criteria to evaluate the effectiveness of compath decomposition: run time, memory requirements, and solution accuracy. First, we describe the software implementation of compath decomposition, then we consider test results.

The software prototype was written in ANSI-C and is called COMET, which stands for COMpath nETwork decomposition. The steps of COMET are summarized in Figure 2. In step 2, $R[(p_1 c, \ldots, p_K c)]$ indicates the value of the primal heuristic when $\tilde{c} = (p_1 c, \ldots, p_K c)$. In step 9, the

set G includes ϵ-supergradients of the Lagrangian $L(\pi)$. In step 11, the direction d is projected to become feasible. In step 13, the iterate is projected to become feasible for the step size α. The dual bound is updated in step 14, the primal bound is updated in step 15, and the estimated solution value is updated in step 16.

COMET uses the following parameters: $\beta = 1.0$, $\epsilon_1 = 5$, $2.5 \leq \epsilon_2 \leq 4000$, and $\epsilon_3 = 1.0 \times 10^{-8}$. Several small changes to the algorithm improve the run time of COMET. First, we restrict the set G to supergradients obtained in recent iterations. In other words, $G \subseteq \{g_{(n-k+1)}, \ldots, g_{(n-1)}, g_{(n)}\}$ for some constant k. Second, we do two kinds of simple preprocessing on the graph. One preprocessing step collapses nodes with one in-arc and one out-arc. The other preprocessing step eliminates redundant capacity constraints. For instance, we only need one constraint $x_a^{k_1} \leq u_a^{k_1}$ if the nonanticipativity constraints imply $x_a^{k_1} = x_a^{k_1+1} = \ldots = x_a^{k_2}$. Third, the primal bound is not updated during every iteration. Instead, step 15 is executed only when the dual value z_d is close to the estimated solution $\hat{z}$.

We compared COMET with linear programming solutions from CPLEX 4.0 [4]. We used AMPL [5] to generate the linear programs for CPLEX. We achieved the best run times by disabling AMPL's preprocessing and using the dual simplex method with CPLEX's default settings. We also tested a commercial interior point algorithm, but the factorizations consumed so much memory that it was only able to solve our smallest problems.

We tested all software on an IBM RS/6000 model 590 workstation running AIX 3.2.5 with 256 MB of RAM and 512 MB of swap space (virtual memory). This machine is benchmarked at SPECint95 = 3.33 and SPECfp95 = 10.4. This system cannot measure precisely the maximum amount of memory used by a particular program. However, we can specify a memory limit and observe whether the program runs successfully. We used this to determine lower bounds for CPLEX's memory requirements and upper bounds for COMET's memory requirements. This allows us to

compute a minimum ratio for the memory savings achievable by COMET. For COMET, we found a memory limit where COMET ran successfully. For CPLEX, we first generated an MPS file using AMPL. Then we called CPLEX separately and found a memory limit where CPLEX failed to solve the LP. By using this two-step procedure, we only measured the memory requirements for solving the linear program. Since the CPLEX memory estimate requires additional steps, we ran it separately from the CPU time measurement.

The test problems are summarized in Table 2. There are two groups of test problems. The

Problem	Nodes	Arcs	Comm	Row	Col	NFC	NAC	Non-0	Density
atl6	565	1078	76	353002	491568	257850	95152	1189400	6.9e-06
dca5	732	1332	130	506000	865800	476100	29900	1830400	4.2e-06
den3	878	1640	110	356987	541200	289887	67100	251790	1.3e-06
mco6	707	1304	66	352788	516384	280254	72534	1196448	6.6e-06
sea4	859	1605	73	291600	468660	251012	40588	1031928	7.6e-06
(10,63)	30	49	15	54705	46305	28665	26040	149415	5.9e-05
(10,97)	30	49	15	86495	71295	44135	42360	234585	3.8e-05
(20,50)	80	139	45	255000	312750	180750	74250	807750	1.0e-05
(20,65)	80	139	45	334335	406575	234975	99360	1055745	7.8e-06
(30,30)	130	229	75	355800	515250	293250	62550	1211850	6.6e-06
(30,50)	130	229	75	612950	858750	488750	124200	2059650	3.9e-06

Table 2: Test Problems

problems in the first group are similar to the problems in [7]. These problems are designated by three letters and a number, where the letters denote an arrival airport and the number specifies the number of scenarios. These may be considered small instances of real-world problems. The second group of problems, designated by (T, K), are fictional problems with three source cities and one destination. T represents the number of time periods and K represents the number of scenarios. These problems are similar to the example picture in [7]. These are clearly toy-sized problems.

In Table 2, Comm represents the number of commodities, and Nodes and Arcs represent the number of nodes and arcs in the graph $\mathcal{G}$. The table includes all nodes and arcs prior to preprocessing. The remaining data describe the linear program (1). Row, Col, and Non-0 specify the

rows, columns, and nonzeros in the LP matrix. NFC and NAC denote the number of network flow constraints and nonanticipativity constraints respectively. Density denotes the ratio of nonzeros to total elements in the LP matrix.

COMET is far more memory efficient than CPLEX, as reported in Table 3. CPLEX was

Problem	CPLEX lower bound	COMET upper bound	Memory Ratio
atl6	225 MB	1 MB	> 225.0
dca5		3 MB	
den3	210 MB	1 MB	> 210.0
mco6	240 MB	1 MB	> 240.0
sea4	200 MB	1 MB	> 200.0
(10,63)	30 MB	1 MB	> 30.0
(10,97)	45 MB	1 MB	> 45.0
(20,50)	155 MB	3 MB	> 51.6
(20,65)	125 MB	3 MB	> 41.6
(30,30)	235 MB	3 MB	> 78.3
(30,50)		3 MB	
average			> 124.6

Table 3: Memory Bounds on RS/6000

not able to solve either dca5 or (30,50) due to memory limitations. When using COMET, the remaining problems have a dramatic reduction in memory use. The lower memory requirements result in a better "wall-clock" time for COMET since the operating system does less paging of virtual memory. They also suggest that only COMET can solve these problems within the standard memory configurations of a desktop PC.

The run times for the test problems are given in Table 4. If we exclude the two problems that CPLEX could not solve, COMET is still more than ten times faster than CPLEX. Table 4 also contains the total number of dual iterations and the values of ϵ_2, which is used to select the ϵ-supergradients.

Problem	CPLEX	COMET	Itns	ϵ_2
atl6	0:04:28	0:00:37	5000	100.0
dca5		0:00:40	3000	1000.0
den3	0:03:29	0:00:21	1750	100.0
mco6	0:03:35	0:00:26	1750	10.0
sea4	0:03:14	0:00:37	5000	1000.0
(10,63)	0:00:38	0:00:04	1250	2.0
(10,97)	0:01:04	0:00:05	1250	2.0
(20,50)	0:07:51	0:00:36	3000	2.5
(20,65)	0:12:49	0:00:45	3000	2.5
(30,30)	0:13:24	0:01:08	4000	15.0
(30,50)		0:02:03	5000	15.0
total	0:50:32	0:07:22		

Table 4: CPU Times on RS/6000 model 590

The solution accuracy for COMET is given in Table 5. The primal gaps are mostly caused by

	LP	COMET					
Problem	optimum	Primal	Diff	% Diff	Dual	Diff	% Diff
atl6	29674.3	29865.6	191.3	0.64%	29534.0	140.3	0.47%
dca5	5901.5	68009.7	45.6	0.07%	67174.7	789.4	1.16%
den3	5901.5	5901.5	0.0	0.00%	5899.0	2.5	0.04%
mco6	1284.8	1284.8	0.0	0.00%	1284.7	0.0	0.00%
sea4	47033.4	49642.7	2609.3	5.55%	46229.9	803.5	1.71%
(10,63)	72.1	75.5	3.4	4.69%	70.6	1.5	2.06%
(10,97)	72.2	75.7	3.5	4.85%	70.7	1.5	2.05%
(20,50)	433.7	461.5	27.8	6.40%	423.7	10.0	2.32%
(20,65)	432.8	462.1	29.4	6.79%	424.6	8.1	1.88%
(30,30)	1168.2	1197.3	29.1	2.49%	1145.0	23.2	1.99%
(30,50)	1168.2	1181.9	40.2	3.52%	1112.1	29.5	2.59%
average				3.18%			1.48%

Table 5: Solution Accuracy for COMET

integrality gaps in the problems. Since the dual optimization is an iterative procedure, we could improve the dual solutions by increasing the number of nonsmooth optimization steps. However, this demonstrates that compath decomposition finds a near-optimal primal integral solution and a near-optimal dual solution using less memory and in less time than CPLEX can find an optimal LP solution. With COMET, these problems can be solved well within the memory limitations of our computer. However, solving these problems with CPLEX puts our machine at or beyond its

memory capacity.

For comparison, consider some single commodity test problems. These problems are summarized in Table 6. The single commodity test problems are also similar to the air traffic problems in [7].

Problem	Nodes	Arcs	Row	Col	NFC	NAC	Non-0	Density
atl150	565	1078	138423	161700	84750	53673	430746	1.9e-05
atl250	565	1079	236852	269500	141250	95602	730204	1.1e-05
dca150	732	1332	138132	199800	109800	28332	456264	1.7e-05
den150	878	1640	199464	246000	131700	67764	627528	1.3e-05
den250	878	1640	340869	410000	219500	121369	1062738	7.6e-06
mco150	792	1515	163136	227250	118800	44336	543172	1.5e-05
mco250	792	1515	275706	378750	198000	77706	912912	8.7e-06
sea150	859	1605	197658	240750	128850	68808	619116	1.3e-05

Table 6: Single Commodity Test Problems

Note that mco50 is different from the other mco problems, though, since the smaller graph was infeasible for the larger scenario samples.

Again, COMET is more memory efficient than CPLEX, as reported in Table 7. For the single commodity problems, the run times and solution accuracy are reported in Tables 8 and 9, respectively. With single commodity problems, COMET finds a near-optimal primal integral solution and a near-optimal dual solution in about the same amount of time as CPLEX finds an LP optimal solution.

7 Conclusions

COMET can find a near-optimal primal integral solution and a near-optimal dual solution using much less memory and in less time than commercial software can find an optimal LP solution. This memory reduction enables us to solve problems that were too large for commercial LP software. For multicommodity models, the memory and time savings are significant.

Problem	CPLEX lower bound	COMET upper bound	Memory Ratio
atl150	100 MB	5 MB	$>$ 20.0
atl250	155 MB	15 MB	$>$ 10.3
dca150	100 MB	20 MB	$>$ 5.0
den150	130 MB	5 MB	$>$ 26.0
den250	220 MB	25 MB	$>$ 8.8
mco150	110 MB	15 MB	$>$ 7.3
mco250	190 MB	20 MB	$>$ 9.5
sea150	135 MB	15 MB	$>$ 9.0
average			$>$ 9.9

Table 7: Memory Bounds for Single Commodity Problems on RS/6000

Problem	CPLEX	COMET	Itns	ϵ_2
atl150	0:05:17	0:05:37	3000	250
atl250	0:10:47	0:11:08	3500	400
dca150	0:15:26	0:15:02	4000	1250
den150	0:12:09	0:08:20	1750	100
den250	0:12:00	0:13:04	1750	100
mco150	0:06:04	0:06:12	1750	10
mco250	0:11:04	0:10:13	1750	10
sea150	0:20:12	0:10:43	3500	1500
total	1:32:59	1:20:19		

Table 8: CPU Times on RS/6000 model 590 for Single Commodity Problems

The slowest component of the COMET implementation is the dual optimization. Over 90% of the computation time is spent performing the nonsmooth optimization. It is possible that a better implementation could improve the dual convergence. One way to improve the direction generation would be to have a dynamically changing ϵ value when selecting ϵ-subgradients. Another is to try the bundle-trust methods in [23].

For many applications, the key problem is to generate good primal solutions quickly. We observed that the greedy primal heuristic gives good primal solutions even with the initial cost vector $\tilde{c} = (p_1 c, \ldots, p_K c)$. Since this starting solution was quite good, there was little improvement at the end of the COMET algorithm. We could devise a pure-primal heuristic that does a local

Problem	LP optimum	COMET Primal	Diff	% Diff	Dual	Diff	% Diff
atl150	25050.2	26202.0	1151.8	4.60%	24267.7	782.5	3.12%
atl250	26538.6	27695.8	1157.2	4.36%	25590.5	948.1	3.57%
dca150	61879.1	63677.3	1798.2	2.91%	59767.3	2111.8	3.41%
den150	17406.0	17503.1	97.1	0.56%	17223.2	182.8	1.05%
den250	19009.7	19133.5	123.9	0.65%	18803.0	206.7	1.09%
mco150	2751.5	2769.3	17.8	0.65%	2695.5	56.0	2.03%
mco250	3059.9	3076.4	16.5	0.54%	2979.6	80.4	2.63%
sea150	77971.5	81568.3	3596.8	4.61%	75168.4	2803.0	3.59%
average				2.36%			2.56%

Table 9: Solution Accuracy for Single Commodity Problems with COMET

search on $\tilde{c}$. However, without a lower bound, it would be difficult to tell when the primal solution was close to the optimal objective value.

Finally, the dual initialization procedure worked very well for this problem. Although postponing the recomputation procedure degrades the estimate, the difference quickly becomes insignificant after applying the dual optimization procedure. This idea of modifying a primal algorithm to obtain approximate marginal values may have other applications in combinatorial optimization.

References

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

[2] J. R. Birge and F. Louveaux, *Introduction to Stochastic Programming*, Springer-Verlag, New York, 1997.

[3] R. K.-M. Cheung and W. B. Powell, An Algorithm for Multistage Dynamic Networks with Random Arc Capacities, with an Application to Dynamic Fleet Management, *Operations Research* 44, 951–963, 1996.

[4] CPLEX Optimization, Inc., Incline Village NV, *CPLEX*, 4.0 edition, 1995.

[5] R. Fourer, D. M. Gay, and B. W. Kernighan, *AMPL: A Modeling Language for Mathematical Programming*, The Scientific Press (now part of Wadsworth Publishing), South San Francisco CA, 1993.

[6] L. F. Frantzeskakis and W. B. Powell, A Successive Linear Approximation Procedure for Stochastic, Dynamic Vehicle Allocation Problems, *Transportation Science* 24, 40–57, 1990.

[7] G. D. Glockner, Effects of Air Traffic Congestion Delays Under Several Flow Management Policies, *Transportation Research Record* 1517, 29–36, 1996.

[8] G. D. Glockner, *Dynamic Network Flow with Uncertain Arc Capacities*, PhD thesis, Georgia Institute of Technology, Atlanta GA, 1997.

[9] G. D. Glockner and G. L. Nemhauser, Dynamic Network Flow with Uncertain Arc Capacities: Formulation and Problem Structure, Technical Report 96-08, Logistics Engineering Center, Georgia Institute of Technology, Atlanta GA, 1996, to appear in *Operations Research*.

[10] F. Glover, D. Klingman, and N. V. Phillips, *Network Models in Optimization and Their Applications in Practice*, John Wiley and Sons, New York, 1992.

[11] M. Held, P. Wolfe, and H. P. Crowder, Validation of Subgradient Optimization, *Mathematical Programming* 6, 62–88, 1974.

[12] W. C. Jordan and M. A. Turnquist, A Stochastic, Dynamic Network Model for Railroad Car Distribution, *Transportation Science* 17, 123–145, 1983.

[13] P. Kall and S. W. Wallace, *Stochastic Programming*, John Wiley and Sons, New York, 1994.

[14] C. Lemaréchal, Nondifferntiable Optimization, in G. L. Nemhauser, A. H. G. Rinnooy Kan, and M. J. Todd, eds., *Optimization*, volume 1 of *Handbooks in Operations Research and Management Science*, pp. 529–572, North-Holland, New York, 1989.

[15] M. Minoux, *Mathematical Programming: Theory and Algorithms*, John Wiley and Sons, New York, 1986.

[16] J. M. Mulvey and H. Vladimirou, Stochastic Network Optimization Models for Investment Planning, *Annals of Operations Research* 20, 187–217, 1989.

[17] W. B. Powell and R. K.-M. Cheung, A Network Recourse Decomposition Method for Dynamic Networks with Random Arc Capacities, *Networks* 24, 369–384, 1994.

[18] W. B. Powell and R. K.-M. Cheung, Stochastic Programs over Trees with Random Arc Capacities, *Networks* 24, 161–175, 1994.

[19] W. B. Powell, Y. Sheffi, K. S. Nickerson, K. Butterbaugh, and S. Atherton, Maximizing Profits for North American Van Lines' Truckload Division: A New Framework for Pricing and Operations, *Interfaces* 18, 21–41, 1988.

[20] O. Richetta and A. R. Odoni, Dynamic Solution to the Ground-Holding Problem in Air Traffic Control, *Transportation Research Part A: Policy and Practice* 28, 167–185, 1994.

[21] R. T. Rockafellar, *Convexity Analysis*, Princeton University Press, Princeton NJ, 1970.

[22] R. T. Rockafellar and R. J.-B. Wets, Scenarios and Policy Aggregation in Optimization Under Uncertainty, *Mathematics of Operations Research* 16, 119–147, 1991.

[23] H. Schramm and J. Zowe, A Version of the Bundle Idea for Minimizing a Nonsmooth Function: Conceptual Idea, Convergence Analysis, Numerical Results, *SIAM Journal on Optimization* 2, 121–152, 1992.

[24] N. Z. Shor, *Minimization Methods for Nondifferentiable Functions*, Springer-Verlag, Berlin, 1985.

[25] P. B. Vranas, D. J. Bertsimas, and A. R. Odoni, Dynamic Ground-Holding Policies for a Network of Airports, *Transportation Science* 28, 275–291, 1994.

[26] P. B. Vranas, D. J. Bertsimas, and A. R. Odoni, The Multi-Airport Ground-Holding Problem in Air Traffic Control, *Operations Research* 42, 249–261, 1994.

[27] A. P. M. Wagelmans, *Sensitivity Analysis in Combinatorial Optimization*, PhD thesis, Erasmus University, Rotterdam, The Netherlands, 1990.

[28] D. Wedelin, An Algorithm for Large Scale 0-1 Integer Programming with Application to Airline Crew Scheduling, *Annals of Operations Research* 57, 283–301, 1995.

[29] R. J.-B. Wets, The Aggregation Principle in Scenario Analysis and Stochastic Programming, in S. W. Wallace, ed., *Algorithms and Model Formulations in Mathematical Programming*, pp. 91–113, Springer-Verlag, New York, 1989.