SOLVING THE CONVEX COST INTEGER DUAL NETWORK FLOW PROBLEM

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

Dorit S. Hochbaum
Department of IE and OR, and
Haas School of Management
University of California
Berkeley, CA 94720, USA
dorit@hochbaum.ieor.berkeley.edu

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

(Revised July 28, 2001; April 3, 2002; August 9, 2002)

SOLVING THE CONVEX COST INTEGER DUAL NETWORK FLOW PROBLEM[*]

Ravindra K. Ahuja[1], Dorit S. Hochbaum[2], and James B. Orlin[3]

ABSTRACT

In this paper, we consider an integer convex optimization problem where the objective function is the sum of separable convex functions (that is, of the form $\sum_{(i,j)\in Q} \bar{F}_{ij}(w_{ij}) + \sum_{i\in P} \bar{B}_i(\mu_i)$), the constraints are similar to those arising in the dual of a minimum cost flow problem (that is, of the form $\mu_i - \mu_j \le w_{ij}$, $(i, j) \in Q$), with lower and upper bounds on variables. Let n = |P|, m = |Q|, and U be the largest magnitude in the lower and upper bounds of variables. We call this problem *the convex cost integer dual network flow problem*. In this paper, we describe several applications of the convex cost integer dual network flow problem arising in a dial-a-ride transit problem, inverse spanning tree problem, project management, and regression analysis. We develop network flow based algorithms to solve the convex cost integer dual network flow problem. We show that using the Lagrangian relaxation technique, the convex cost integer dual network flow problem can be transformed into a convex cost primal network flow problem where each cost function is a piecewise linear convex function with integer slopes. Its special structure allows the convex cost primal network flow problem to be solved in $O(nm \log(n^2/m) \log(nU))$ time. This time bound is the same time needed to solve the minimum cost flow problem using the cost-scaling algorithm, and is also is best available time bound to solve the convex cost integer dual network flow problem.

[*] An earlier version of the paper appeared in Ahuja, Hochbaum, Orlin [1999].
[1] Department of Industrial and Systems Engineering, University of Florida, Gainesville, FL 32611, USA.
[2] Department of IE and OR and Haas School of Management, University of California, Berkeley, CA 94720, USA.
[3] Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA 02139, USA.

1. INTRODUCTION

We consider the following mathematical programming problem in this paper:

$$\text{Minimize } \sum_{(i,j)\in Q} \overline{F}_{ij}(w_{ij}) + \sum_{i\in P} \overline{B}_i(\mu_i) \tag{1a}$$

subject to

$$\mu_i - \mu_j \le w_{ij} \quad \text{for all } (i, j) \in Q, \tag{1b}$$

$$l_{ij} \le w_{ij} \le u_{ij} \quad \text{for all } (i, j) \in Q, \tag{1c}$$

$$l_i \le \mu_i \le u_i \quad \text{for all } i \in P, \tag{1d}$$

$$w_{ij} \text{ is integer for all } (i, j) \in Q, \text{ and } \mu_i \text{ is integer for all } i \in P, \tag{1e}$$

where $P = \{1, 2, \dots, n\}$ is a set of n numbers, $Q \subseteq P \times P$, and l_i, u_i, l_{ij}, and u_{ij} are specified finite integers. We permit the lower bounds l_i and l_{ij} to take negative values. Let $m = |Q|$. In this problem, the μ_i's and w_{ij}'s are decision variables, $\overline{F}_{ij}(w_{ij})$ is a convex function of w_{ij} for every $(i, j) \in Q$, and $\overline{B}_i(\mu_i)$ is also a convex function of μ_i for every $i \in P$. Let $U = \max[\max\{u_{ij} - l_{ij}: (i, j) \in Q\}, \max\{u_i - l_i : i \in P\}]$. We assume that $U < \infty$.

We call problem (1) the *convex cost integer dual network flow problem* (or simply the *dual network flow problem*), because the dual of the minimum cost flow problem is easily transformed into a special case of (1). The dual network flow problem and its special cases arise in various application settings, including multi-product multistage production/inventory systems, project scheduling, machine-scheduling with precedence constraints, dial-a-ride transit problems, inverse optimization, and isotone regression. We describe some of these applications in Section 5 and provide references for other applications.

It is well known (see, for example, Murty [1985]) that (1) can be transformed into a linear programming problem by assuming (without loss of generality) that each of the convex functions is linear between successive integers, and then introducing a separate variable for each linear segment in the functions $F_{ij}(w_{ij})$ and $B_i(\mu_i)$. It follows from the integrality of the breakpoints (that is, those points where the slopes of the functions $F_{ij}(w_{ij})$ and $B_i(\mu_i)$ change) that there always exist an optimal solution of (1) that is integer. Hence this assumption:

Assumption 1. *We assume without loss of generality that each of the convex functions is linear between successive integers. We also assume that each convex function can be evaluated in O(1) steps.*

We point out that this assumption does not increase the problem size because we handle the objective function (1a) implicitly. We assume that each of the functions $\overline{F}_{ij}(w_{ij})$ (or $\overline{B}_i(\mu_i)$) can be evaluated in O(1) time for a given value of w_{ij} (or μ_i) using an oracle function and is independent of the number of segments in the function. In fact, the running times stated below are functions of the number of nodes n, the number of arcs m, and log U, and they do not involve the size of the data needed to specify the cost functions.

In (1) we permit lower and upper bounds on the variables w_{ij} and μ_i. However, our algorithm presumes that no such bound exists. We can transform (1) to eliminate the bounds.

Assumption 2: *There are no lower and upper bound constraints on the variables w_{ij} and μ_i.* We can easily satisfy this assumption by modifying $\overline{F}_{ij}(w_{ij})$ and $\overline{B}_i(\mu_i)$ as follows:

$$F_{ij}(w_{ij}) = \begin{cases} \overline{F}_{ij}(u_{ij}) + M(w_{ij} - u_{ij}) & \text{for } w_{ij} > u_{ij} \\ \overline{F}_{ij}(w_{ij}) & \text{for } l_{ij} \le w_{ij} \le u_{ij}, \\ \overline{F}_{ij}(l_{ij}) - M(w_{ij} - l_{ij}) & \text{for } w_{ij} < l_{ij} \end{cases} \tag{2a}$$

$$B_i(\mu_i) = \begin{cases} \overline{B}_i(u_i) + M(\mu_i - u_i) & \text{for } \mu_i > u_i \\ \overline{B}_i(\mu_i) & \text{for } l_i \le \mu_i \le u_i, \\ \overline{B}_i(l_i) - M(\mu_i - l_i) & \text{for } \mu_i < l_i \end{cases} \tag{2b}$$

where M is a sufficiently large number. We choose M such that in any solution (w, μ) that satisfies (1b) but violates (1c) or (1d) cannot be an optimal solution of (1). Any value of $M > L_f - L_b$ will suffice, where L_f denotes a feasible objective function value and L_b denotes a lower bound on the objective function value. To determine L_f, we can solve: Maximize $\sum_{(i,j)\in Q} \overline{F}_{ij}(w_{ij}) + \sum_{i \in P} \overline{B}_i(\mu_i)$, subject to (1c), (1d), and (1e). Since this problem is separable in each decision variable and each variable takes integer values, we can determine the optimal value of each decision variable in O(log U) time using any search method. Hence, L_f can be determined in a total of O((m+n) log U) time. To determine L_b, we can solve: Minimize $\sum_{(i,j)\in Q}\bar{F}_{ij}(w_{ij}) + \sum_{i\in P}\bar{B}_i(\mu_i)$, subject to (1c), (1d), and (1e), which can be also be done in O((m+n) log U) time.

Assumption 3: *Each constraint in (1b) is an equality constraint.* Often in dual network flow problems, the constraints are of the form of inequalities: $\mu_i - \mu_j \le w_{ij}$ for all $(i, j) \in Q$. In such a case, the contribution to the objective function for variable w_{ij} would not be $F_{ij}(w_{ij})$, but instead would be $\min\{F_{ij}(x) : x \ge w_{ij}\}$ since increasing the value of w_{ij} would keep the inequality $\mu_i - \mu_j \le w_{ij}$ satisfied. One can transform a problem with inequalities to one with equalities in polynomial time by letting $E_{ij}(w_{ij}) = \text{minimum}\{F_{ij}(x) : x \ge w_{ij}\}$, and noting that $E_{ij}(w_{ij})$ is convex. Assuming that $F_{ij}(w_{ij})$ can be evaluated in O(1) steps for each integral w_{ij}, it follows that $E_{ij}(w_{ij})$ can be evaluated in O(log U) steps for each integer w_{ij}. Moreover, we can do better and evaluate it in O(1) time after some preprocessing has been done. Suppose that we choose α_{ij} such that $F_{ij}(\alpha_{ij}) = \min\{F(w_{ij}): l_{ij} \le w_{ij} \le u_{ij}\}$. Then, $E_{ij}(w_{ij}) = F_{ij}(\alpha_{ij})$ if $x_{ij} \le \alpha_{ij}$, and $E_{ij}(w_{ij}) = F_{ij}(w_{ij})$ if $x_{ij} > \alpha_{ij}$. We can determine α_{ij} in O(log U) time for any $(i, j) \in Q$, and in O(m log U) time overall. Once α_{ij}'s are determined, we can evaluate $E_{ij}(w_{ij})$ in O(1) time for any w_{ij}.

We will henceforth assume that the transformation described above to satisfy Assumptions 1, 2, and 3 have been made. These transformations give us the following formulation:

$$\text{Minimize } \sum_{(i,\, j)\in Q} F_{ij}(w_{ij}) \;+\; \sum_{i\in P} B_i(\mu_i) \tag{3a}$$

subject to

$$\mu_i - \mu_j = w_{ij} \qquad \text{for all } (i, j) \in Q. \tag{3b}$$

In this paper, we develop an O(nm log(n^2/m) log(nU)) time algorithm to solve (1) which improves the previous best time bound of O(nm log n log(nU)) due to Karzanov and McCormick [1997]. Our algorithm follows from the series of three results, two of which are due to Rockafellar [1984]. However, since our notation is significantly different than that of Rockafellar and we also use some problem specific properties of (1), we develop these results from scratch. These three results are:

(1) The Lagrangian relaxation of (3) obtained by relaxing the constraints in (3b) can be transformed to a network flow problem with nonlinear costs. This result is shown in Section 2.

(2) The network flow problem with nonlinear costs described above is a convex cost network flow problem, that is, the minimum cost flow problem with piecewise linear convex functions where each linear segment has integer slopes. This result is shown in Section 3.

(3) We next adapt the cost-scaling algorithm for the minimum cost flow problem, due to Goldberg and Tarjan [1987], to solve the convex cost network flow problem. Using the integrality of the slopes of the linear segments in the cost functions and some additional properties, we show that the cost-scaling algorithm can solve the convex cost integer dual network flow problem in O(nm log(n^2/m) log(nU)) time. This result is shown in Section 4. This time bound is the same time needed to solve the minimum cost flow problem using the cost-scaling algorithm, and is also is best available time bound to solve the convex cost integer dual network flow problem for all network densities (that is, for all possible values of m/n).

We now briefly survey the related research. Hochbaum and Shanthikumar [1990] showed how to solve the dual network flow problem in polynomial time. In that paper, it was shown that any convex separable minimization in integers over totally unimodular constraint matrix is solvable in polynomial time by O(log U) calls to a linearized version of the problem in binary variables. Recently, Ahuja, Hochbaum and Orlin [2000] demonstrated how to implement the algorithm for the linearized version of the problem (1) by solving a minimum cut problem on an associated graph. The running time of this approach to solve (1) is O((log U) T(n^2, mn^2)), where T(p, q) represents the running time to solve a minimum cut problem on a network with p nodes and q arcs.

The problem of convex optimization over totally unimodular constraints was also addressed by Karzanov and McCormick [1997]. They proposed two approaches by generalizing the *minimum mean cycle canceling method* and the *cancel-and-tighten method* for the minimum cost flow problem developed by Goldberg and Tarjan [1988]. Both of these two approaches apply to the convex cost integer dual network flow problem, which they called the *network cocirculation problem*. The cancel-and-tighten method runs faster than the minimum mean cycling method and its running time for the convex cost integer dual network flow problem is O(nm log n log(nU)). Whereas Karzanov and McCormick's method applies the cancel-and-tighten method due to Goldberg and Tarjan [1988] which proceeds by sending flows along negative cost cycles, our method applies cost scaling algorithm due to Goldberg and Tarjan [1987] which proceeds by pushing flows from nodes with excesses towards nodes with deficits.

Hochbaum [2001a] recently devised an algorithm for a special case of our problem, with the functions $\overline{F}_{ij}(w_{ij})$ restricted to be linear and $w_{ij} \geq 0$. That algorithm has the complexity of solving a minimum cut problem on a graph T(n, m) plus the complexity of finding the integer minima of the convex functions $\overline{B}_i(\mu_i)$, which can be found in O(n log U) for general convex functions, or faster for specific convex functions such as quadratic functions or piecewise linear functions.

2. Transformation to a Network Flow Problem

We use the Lagrangian relaxation technique of Rockafellar [1984] to solve the dual network flow problem. He showed that the Lagrangian multiplier problem is a minimum convex cost network flow problem. We review his approach in this section.

We dualize the constraints (3b) using the vector x, obtaining the following *Lagrangian subproblem*:

$$L(x) = \min_{w,\mu} \sum_{(i,j)\in Q} F_{ij}(w_{ij}) + \sum_{i\in P} B_i(\mu_i) - \sum_{(i,j)\in Q} (w_{ij} + \mu_j - \mu_i)x_{ij}, \tag{4}$$

where x_{ij} is the Lagrangian multiplier associated with the constraint $\mu_i - \mu_j = u_{ij}$. It is easy to show that

$$\sum_{(i,j)\in Q} (\mu_j - \mu_i)\, x_{ij} = \sum_{i\in P} \mu_i\Big(\sum_{\{j:(j,i)\in Q\}} x_{ji} - \sum_{\{j:(i,j)\in Q\}} x_{ij}\Big) = \sum_{i\in P} x_{i0}\mu_i, \tag{5}$$

where $x_{i0} = \sum_{\{j:(j,i)\in Q\}} x_{ji} - \sum_{\{j:(i,j)\in Q\}} x_{ij}$ for each $i \in P$. Substituting (5) into (4) yields

$$L(x) = \min_{w,\mu} \sum_{(i,j)\in Q} \{ F_{ij}(w_{ij}) - x_{ij}\, w_{ij}\} + \sum_{i\in P}\{ B_i(\mu_i) - x_{i0}\mu_i\}, \tag{6a}$$

subject to

$$x_{i0} = \sum_{\{j:(j,i)\in Q\}} x_{ji} - \sum_{\{j:(i,j)\in Q\}} x_{ij} \qquad \text{for all } i \in P. \tag{6b}$$

We will now simplify the Lagrangian subproblem (6). We define a directed network G = (N, A) with node set N and arc set A. The node set N contains a node i for each element $i \in P$ and an extra node,

node 0. The arc set A contains an arc (i, j) for each (i, j) $\in$ Q and an arc (i, 0) for each i $\in$ P. For each arc (i, 0), i $\in$ P, we let $w_{i0} = \mu_i$, $l_{i0} = l_i$, $u_{i0} = u_i$, and $F_{i0}(w_{i0}) = B_i(\mu_i)$. Observe that |N| = n + 1, and |A| = m + n. In terms of these notations, the Lagrangian subproblem (6) can be restated as follows:

$$L(x) = \min_{w} \sum_{(i, j)\in A} \{F_{ij}(w_{ij}) - x_{ij}w_{ij}\} \tag{7a}$$

subject to

$$\sum_{\{j:(j,i)\in A\}} x_{ji} - \sum_{\{j:(i,j)\in A\}} x_{ij} = 0 \qquad \text{for all } i \in N. \tag{7b}$$

Since each $F_{ij}(w_{ij})$ is a convex function, $F_{ij}(w_{ij}) - x_{ij}w_{ij}$ is also a convex function of w_{ij} for a given value of x_{ij}. (When we mention of a specified value of x, we mean that we specify x_{ij} for each (i, j) $\in$ Q, and the remaining x_{ij} for each (i, j) $\in$ A\Q are determined using (7b).) For a given value of x, each term of $\min_{w} \sum_{(i, j)\in A} \{F_{ij}(w_{ij}) - x_{ij}w_{ij}\}$ can be optimized separately. Consequently, in order to determine L(x) for a given x, we need to determine $\min\{F_{ij}(w_{ij}) - x_{ij}w_{ij}: w_{ij} \text{ integer}\}$ for each (i, j) $\in$ A.

Let $H_{ij}(x_{ij}) = \min\{F_{ij}(w_{ij}) - x_{ij}w_{ij} : w_{ij} \text{ integer}\}$ for each (i, j) $\in$ A. In terms of $H_{ij}(x_{ij})$, (7) can be restated as:

$$L(x) = \sum_{(i, j)\in A} H_{ij}(x_{ij}) \text{ subject to (7b).} \tag{8}$$

We now focus on solving the *Lagrangian Multiplier Problem,* which is to determine the value of x for which the Lagrangian function L(x) attains the highest objective function value. The Lagrangian multiplier problem is to determine x* such that

$$L(x^*) = \max L(x) = \max \sum_{(i, j)\in A} H_{ij}(x_{ij}) \text{ subject to (7b),} \tag{9a}$$

which is a network flow problem in terms of the arc flow vector x (which is the vector of decision variables) with nonlinear cost functions and with no upper or lower bounds on arc flows x.

The following well-known theorem establishes a connection between the Lagrangian multiplier problem and the dual network flow problem.

Theorem 1. *Let x* be an optimal solution of the Lagrangian multiplier problem (9). Then L(x*) equals the optimal objective function value of the dual network flow problem (3).*

Proof: The problem (9a) can be transformed to a linear programming problem by introducing a separate variable for each linear segment in the cost functions $H_{ij}(x_{ij})$. Then, Theorem 1 follows from a well-known result in the theory of Lagrangian relaxation (see, for example, Ahuja, Magnanti and Orlin [1993], Theorem 16.8). ♦

We describe in Section 4 how we can efficiently use the optimal solution of (3) to construct an optimal solution of (1). We now discuss the time taken to transform the problem (1) into (3). It takes $O((m+n) \log U)$ time to satisfy Assumption 2 and to go from (1) to (2). It takes $O(m \log U)$ time to satisfy Assumption 3 and to go from (2) to (3). As we will see later, this time is dominated by the time taken by other steps in our algorithm.

3. Properties of the Function $H_{ij}(x_{ij})$

In this section, we study properties of the function $H_{ij}(x_{ij})$. We shall use these properties in the next section to develop an efficient cost-scaling algorithm to solve the Lagrangian multiplier problem (9a).

We first study how to compute $H_{ij}(x_{ij})$ for a given value of x_{ij}. Recall that $H_{ij}(x_{ij}) = \min\{F_{ij}(w_{ij}) - x_{ij}w_{ij} : w_{ij} \text{ integer}\}$. There are three cases to consider:

(i) **Case 1**: $x_{ij} > M$. It follows from (2a) that if $w_{ij} > M$, then increasing w_{ij} further increases $F_{ij}(w_{ij})$ at a rate of M per unit increase in w_{ij}. But increasing w_{ij} decreases $-x_{ij}w_{ij}$ at a rate strictly greater than M (because $x_{ij} > M$). Consequently, $F_{ij}(w_{ij}) - x_{ij}w_{ij}$ strictly decreases if we increase w_{ij} beyond M. Hence, $F_{ij}(w_{ij}) - x_{ij}w_{ij}$ approaches $-\infty$ as w_{ij} approaches ∞. Therefore, $H_{ij}(x_{ij}) = -\infty$ for $x_{ij} > M$.

(ii) **Case 2**: $x_{ij} < -M$. An analysis similar to that in Case 1 yields that $H_{ij}(x_{ij})$ approaches $-\infty$ as w_{ij} approaches $-\infty$. Therefore, $H_{ij}(x_{ij}) = -\infty$ for $x_{ij} < -M$.

(iii) **Case 3**: $-M \le x_{ij} \le M$. In this case, we will show that $F_{ij}(w_{ij}) - x_{ij}w_{ij}$ will achieve its minimum for some w_{ij} satisfying $l_{ij} \le w_{ij} \le u_{ij}$ (where l_{ij} and u_{ij} are as defined in Section 1). To see this, observe that for $w_{ij} > u_{ij}$, $F_{ij}(w_{ij})$ increases at a rate of M per unit increase in w_{ij}, and $-x_{ij}w_{ij}$ decreases at a rate no more than M (because $x_{ij} \le M$); consequently, $F_{ij}(w_{ij}) - x_{ij}w_{ij}$ increases as

w_{ij} increases. A similar argument applies when $w_{ij} < l_{ij}$. Hence $H_{ij}(x_{ij}) = \min\{F_{ij}(w_{ij}) - x_{ij}w_{ij}: l_{ij} \le w_{ij} \le u_{ij}$ and w_{ij} integer$\}$. Since $F_{ij}(w_{ij})$ is a piecewise linear convex function, $F_{ij}(w_{ij}) - x_{ij}w_{ij}$ is also a piecewise linear convex function. Therefore, we can find $H_{ij}(x_{ij}) = \min\{F_{ij}(w_{ij}) - x_{ij}w_{ij}: l_{ij} \le w_{ij} \le u_{ij}$ and w_{ij} integer$\}$ by performing binary search and considering integer values of w_{ij} in the interval $[l_{ij}, u_{ij}]$, which requires O(log U) time. Thus, in this case, $H_{ij}(x_{ij})$ can be computed in O(log U) time. (Here, as per our assumption, $F_{ij}(w_{ij})$ can be computed in O(1) time for a specified value of w_{ij}.)

Recall that the Lagrangian multiplier problem is to determine a vector x that maximizes $\sum_{(i, j)\in A} H_{ij}(x_{ij})$ subject to (7b). The preceding discussion implies that $H_{ij}(x_{ij}) = -\infty$ when $x_{ij} > M$ or when $x_{ij} < -M$. Thus we can exclude the values $x_{ij} > M$ and $x_{ij} < -M$ when solving the Lagrangian multiplier problem. Alternatively, the Lagrangian multiplier problem can be stated as to maximize $\sum_{(i, j)\in A} H_{ij}(x_{ij})$ subject to (7b) and the following additional redundant constraints:

$$-M \le x_{ij} \le M \quad \text{for all } (i, j) \in A, \tag{9b}$$

where $H_{ij}(x_{ij}) = \min\{F_{ij}(w_{ij}) - x_{ij}w_{ij}: w_{ij}$ integer$\}$. We will next show that $H_{ij}(x_{ij})$ is a piecewise linear concave function of x_{ij}. Our subsequent discussion uses the following property:

Property 1.

(a) *Let $F_{ij}(\theta)$ be a convex function of θ. Then, $F_{ij}(\theta + 1) - F_{ij}(\theta) \le F_{ij}(\theta + 2) - F_{ij}(\theta + 1) \le F_{ij}(\theta + 3) - F_{ij}(\theta + 2) \le \ldots\ldots \le F_{ij}(u_{ij}) - F_{ij}(u_{ij} - 1)$.*

(b) *Let $b_{ij}(\theta) = F_{ij}(\theta + 1) - F_{ij}(\theta)$ for $l_{ij} \le \theta \le u_{ij} - 1$, θ integral. Then, $b_{ij}(\theta) \le b_{ij}(\theta + 1) \le b_{ij}(\theta + 2) \le \ldots \le b_{ij}(u_{ij} - 1)$.*

Property 1(a) directly follows from the convexity of the function $F_{ij}(\theta)$, and Property 1(b) is equivalent to Property 1(a). Since for a given x_{ij}, $F_{ij}(w_{ij}) - w_{ij}x_{ij}$ is a piecewise linear convex function of w_{ij}, $H_{ij}(x_{ij}) = \min\{F_{ij}(w_{ij}) - x_{ij}w_{ij}: w_{ij}$ integer$\} = \min\{F_{ij}(w_{ij}) - x_{ij}w_{ij}: l_{ij} \le w_{ij} \le u_{ij}$ and w_{ij} integer$\} = \min\{F_{ij}(l_{ij}) - x_{ij}l_{ij}, F_{ij}(l_{ij}+1) - x_{ij}(l_{ij}+1), F_{ij}(l_{ij}+2) - x_{ij}(l_{ij}+2), \ldots, F_{ij}(u_{ij}) - x_{ij}u_{ij}\}$. Observe that the function $H_{ij}(x_{ij})$ is the lower envelope of the lines $F_{ij}(l_{ij}) - l_{ij}x_{ij}$, $F_{ij}(l_{ij}+1) - (l_{ij}+1)x_{ij}$, $F_{ij}(l_{ij}+2) - (l_{ij}+2)x_{ij}$, … , $F_{ij}(u_{ij}) - u_{ij}x_{ij}$ and hence is a piecewise linear concave function.

Theorem 2. *The function $H_{ij}(x_{ij}) = min\{F_{ij}(w_{ij}) - w_{ij}x_{ij} : w_{ij}$ integer\} is a piecewise linear concave function of x_{ij}, and is described in the following manner:*

$$H_{ij}(x_{ij}) = \begin{cases} F_{ij}(l_{ij}) - l_{ij}x_{ij} & \text{if } -M \le x_{ij} \le b_{ij}(l_{ij}) \\ F_{ij}(l_{ij}+1) - (l_{ij}+1)x_{ij} & \text{if } b_{ij}(l_{ij}) \le x_{ij} \le b_{ij}(l_{ij}+1) \\ \vdots & \\ F_{ij}(q) - qx_{ij} & \text{if } b_{ij}(q-1) \le x_{ij} \le b_{ij}(q) \\ \vdots & \\ F_{ij}(u_{ij}) - u_{ij}x_{ij} & \text{if } b_{ij}(u_{ij}-1) \le x_{ij} \le M \end{cases} \tag{10}$$

Proof. Consider the lines $F_{ij}(\theta) - \theta x_{ij}$ as θ varies from l_{ij} to u_{ij}. The line $F_{ij}(l_{ij}) - l_{ij}x_{ij}$ intersects with the line $F_{ij}(l_{ij}+1) - (l_{ij}+1)x_{ij}$ at $x_{ij} = F_{ij}(l_{ij}+1) - F_{ij}(l_{ij}) = b_{ij}(l_{ij})$. Similarly, the line $F_{ij}(l_{ij}+1) - (l_{ij}+1)x_{ij}$ intersects with the line $F_{ij}(l_{ij}+2) - (l_{ij}+2)x_{ij}$ at $x_{ij} = b_{ij}(l_{ij}+1)$. It follows from Property 1(b) that $b_{ij}(l_{ij}) \le b_{ij}(l_{ij}+1)$. In general, the line $F_{ij}(\theta) - \theta x_{ij}$ intersects with the line $F_{ij}(\theta+1) - (\theta+1)x_{ij}$ at $x_{ij} = b_{ij}(\theta)$ for each $\theta = l_{ij}, l_{ij}+1, l_{ij}+2, \ldots, u_{ij}-1$. This together with the fact that $b_{ij}(l_{ij}) \le b_{ij}(l_{ij}+1) \le b_{ij}(l_{ij}+2) \le \ldots \le b_{ij}(u_{ij})$ establishes the theorem. ♦

It follows from Theorem 2 that the line $F_{ij}(\theta) - \theta x_{ij}$ for a given θ, $l_{ij} \le \theta \le u_{ij}$, is represented in $H_{ij}(x_{ij})$ for $x_{ij} \in [b_{ij}(\theta-1), b_{ij}(\theta)]$ (where $b_{ij}(l_{ij}-1) = -M$ and $b_{ij}(u_{ij}) = M$). If $b_{ij}(l_{ij}-1) < b_{ij}(l_{ij}) < b_{ij}(l_{ij}+1) < b_{ij}(l_{ij}+2) < \ldots < b_{ij}(u_{ij})$, then each line $F_{ij}(\theta) - \theta x_{ij}$ contributes a linear segment of positive length to $H_{ij}(x_{ij})$; in this case, slopes of the linear segments in the function $H_{ij}(x_{ij})$ take all the values from $-l_{ij}$ to $-u_{ij}$ as x_{ij} varies from -M to M. However, if $b_{ij}(\theta-1) = b_{ij}(\theta)$ for some θ, then $F_{ij}(\theta) - \theta x_{ij}$ will contribute just a point (or a line segment of zero length) to $H_{ij}(x_{ij})$. In this case, slopes of the linear segments in the function $H_{ij}(x_{ij})$ will not take all the values from $-l_{ij}$ to $-u_{ij}$ as x_{ij} varies from -M to M; instead at some breakpoints, the slope will change by more than one unit.

We next define the right and left slopes of $H_{ij}(x_{ij})$. For the function $H_{ij}(x_{ij})$, we define its *right slope* at point x_{ij} as $(H_{ij}(x_{ij}+\delta) - H_{ij}(x_{ij}))/\delta$, and its *left slope* at point x_{ij} as $(H_{ij}(x_{ij}) - H_{ij}(x_{ij}-\delta))/\delta$, where δ is a sufficiently small number. In other words, the right slope of $H_{ij}(x_{ij})$ at point x_{ij} equals the slope of the linear segment in $H_{ij}(x_{ij})$ on the right side of the point x_{ij} and the left slope equals the negative of the slope of the linear segment on the left side of the point x_{ij}. It follows from Theorem 2 and our preceding discussion that at $x_{ij} = b_{ij}(\theta)$, the right slope of the function $H_{ij}(x_{ij})$ is at most $-(\theta+1)$ and its left slope is at least $-\theta$.

We have so far transformed the Lagrangian multiplier problem into maximizing a concave cost flow problem. We can alternatively restate this problem as

$$\text{Minimize } \sum_{(i,\, j)\in A} C_{ij}(x_{ij}) \tag{11a}$$

subject to

$$\sum_{\{j:(j,i)\in A\}} x_{ji} - \sum_{\{j:(i,j)\in A\}} x_{ij} = 0 \qquad \text{for all } i \in N, \tag{11b}$$

$$-M \le x_{ij} \le M \qquad \text{for all } (i, j) \in A, \tag{11c}$$

where $C_{ij}(x_{ij}) = -H_{ij}(x_{ij})$. The slopes of the linear segments in $C_{ij}(x_{ij})$ are the negatives of the corresponding slopes in $H_{ij}(x_{ij})$. Since $H_{ij}(x_{ij})$ is a piecewise linear concave function, $C_{ij}(x_{ij})$ is a piecewise linear convex function. The proof of the next lemma follows from this correspondence and the preceding discussion:

Lemma 1. *If $x_{ij} = b_{ij}(\theta)$ for some integer $\theta \in [l_{ij}, u_{ij}-1]$, then the right slope of $C_{ij}(x_{ij})$ is at least $\theta+1$ and its left slope is at most θ. In addition, the right slope of $C_{ij}(x_{ij})$ at $x_{ij} = -M$ is l_{ij}, and the left slope of $C_{ij}(x_{ij})$ at $x_{ij} = M$ is u_{ij}.*

4. Cost-Scaling Algorithm

The problem (11) is a convex cost flow problem in which the cost associated with the flow on arc (i, j) is a piecewise linear convex function containing at most U+1 linear segments. We can transform (11) into a minimum cost flow problem in an expanded network $G' = (N', A')$, and solve it using any minimum cost flow algorithm. In this transformation, for each arc $(i, j) \in A$, we introduce $(u_{ij}-l_{ij}+1)$ arcs in G'-- one arc corresponding to each linear segment, and the costs of these arcs are: l_{ij}, $l_{ij}+1$, $l_{ij}+2$, … , u_{ij}, and the capacities of these arcs, respectively, are $b_{ij}(l_{ij})+M$, $b_{ij}(l_{ij}+1)-b_{ij}(l_{ij})$, $b_{ij}(l_{ij}+2)-b_{ij}(l_{ij}+1)$, …. , $M-b_{ij}(u_{ij}-1)$. Notice that we introduce U+1 arcs in G' for each arc (i, j) in G and the cost of each arc is also bounded by U. It is well known that solving the convex cost flow problem in G is equivalent to solving the minimum cost flow problem in G'. Hence, we can use any minimum cost flow algorithm to solve the minimum cost flow problem in G'. However, since the number of arcs in the network G are O(nU), minimum cost flow algorithms would not in general run in polynomial time. In addition, some minimum cost flow algorithms, such as capacity scaling algorithms, require that the arc capacities are integer (or rational) numbers. In the above transformation, the arc capacities are not necessarily rational numbers (we assumed a convex oracle function, and did not assume rational function values) and in the presence of irrational capacities the capacity scaling algorithm does not finitely converge.

We can, however, use the cost-scaling algorithm due to Goldberg and Tarjan [1987] to solve the minimum cost flow problem in G'. The correctness of the cost-scaling algorithm relies on the fact that all arc costs must be integer. Observe that the minimum cost flow problem in G' has integer costs. Consequently, the cost-scaling algorithm will correctly solve the minimum cost flow problem in G'. However, the running time of the cost-scaling algorithm will not be polynomial in n, m, and log U (because the minimum cost flow problem contains O(nU) arcs). We now describe a modification of the cost-scaling algorithm that can solve the convex cost flow problem in (11) in the same time as needed by the cost-scaling algorithm to solve the minimum cost flow problem. We will subsequently refer to the modified algorithm as the *convex cost-scaling algorithm* and the cost-scaling algorithm for the (linear cost) minimum cost flow problem as the *linear cost-scaling algorithm*.

Our subsequent discussion requires some understanding of the linear cost-scaling algorithm for the minimum cost flow problem. We refer the readers to the paper by Goldberg and Tarjan [1987] or the book of Ahuja, Magnanti and Orlin [1993] for a description of this algorithm. Goldberg and Tarjan had observed that their linear cost-scaling algorithm could be extended to treat convex cost flows at each scaling phase; however, the algorithm is not guaranteed to find an optimal solution of the convex cost flow problem in a polynomially bounded number of scaling phases. (Tarjan [1998] communicated this to one of the co-authors of this paper.) We will show that due to the special structure of (11), we can modify the linear cost-scaling algorithm to obtain an optimal solution of the convex cost flow problem. In addition, we show that the running time of this algorithm to solve (11) is the same as needed to solve the minimum cost flow problem by the linear cost-scaling algorithm.

We now briefly describe the linear cost-scaling algorithm for the minimum cost flow problem and point out the changes we need to make in order to apply it for the convex cost case. We will use the notation given in Ahuja, Magnanti and Orlin [1993]. While describing this algorithm, we will assume that it is applied to the minimum cost flow problem with c_{ij}'s as arc costs, u_{ij}'s as arc capacities, and zero lower bounds on arc costs. The linear cost-scaling algorithm maintains a pseudoflow at each step. A *pseudoflow* x is any function $x : A \rightarrow R$ that satisfies upper and lower bounds on arc flows but may violate the flow balance constraints at nodes. For any pseudoflow x, we define the *imbalance* of node i as $e(i) = \sum_{\{j:(j,i)\in A\}} x_{ji} - \sum_{\{j:(i,j)\in A\}} x_{ij}$ for all $i \in N$. If $e(i) > 0$ for some node i, we refer to node i as an *excess node* and refer to e(i) as the *excess* of node i. We refer to a pseudoflow x with $e(i) = 0$ for all $i \in N$ as a *flow*.

We henceforth assume for notational convenience that for any pair of nodes i and j, either $(i, j) \in A$ or $(j, i) \in A$, but not both. We can easily satisfy this assumption by performing a simple transformation, but the assumption is not needed in any case.

The linear cost-scaling algorithm also maintains a value $\pi(i)$ for each node $i \in N$. We refer to the vector π as a vector of *node potentials*. The cost-scaling algorithm proceeds by constructing and manipulating the residual network G(x) defined as follows with respect to a pseudoflow x. For each arc $(i, j) \in A$, the residual network G(x) contains two arcs (i, j) and (j, i). The arc (i, j) has cost c_{ij} and residual capacity $r_{ij} = u_{ij} - x_{ij}$, and the arc (j, i) has cost $c_{ji} = -c_{ij}$ and residual capacity $r_{ji} = x_{ij}$. The residual network consists *only* of arcs with positive residual capacity.

For a given residual network G(x) and a set of node potentials π, we define the *reduced cost* of an arc (i, j) as $c^{\pi}_{ij} = c_{ij} - \pi(i) + \pi(j)$. For a given value of ε, we call an arc (i, j) *admissible* if $-\varepsilon \le c^{\pi}_{ij} < 0$. A flow or a pseudoflow x is said to be *ε-optimal* for some $\varepsilon \ge 0$ if for some node potentials π, the pair (x, π) satisfies the following *ε-optimality conditions*: $c^{\pi}_{ij} \ge -\varepsilon$ for every arc (i, j) in G(x). The cost-scaling algorithm treats ε as a parameter and iteratively obtains ε-optimal flows for successively smaller values of ε. Initially, $\varepsilon = \max \{c_{ij} : (i, j) \in A\} = U$ and any feasible flow is ε-optimal. The algorithm then performs cost-scaling phases by repeatedly applying an *improve-approximation* procedure that transforms an ε-optimal flow into an ε/2-optimal flow. After O(log(nU)) cost-scaling phases, $\varepsilon < 1/n$ and the algorithm terminates with an optimal flow.

The basic operation in the *improve-approximation* procedure is to select an excess node i and perform *pushes* on admissible arcs emanating from it. The amount pushed on an admissible arc (i, j) is $\min\{r_{ij}, e(i)\}$. If $e(i) < r_{ij}$, then the push is a *nonsaturating* push; otherwise it is a *saturating* push. When node i has no admissible arc emanating from it, then it increases the potential of the node by $\varepsilon/2$; this step is called a *relabel* step. The improve-approximation procedure terminates when there is no excess node (that is, the current pseudoflow is a flow). Hence, there are three major operations per scaling phase in the linear cost-scaling algorithm: saturating pushes, nonsaturating pushes, and relabels. Goldberg and Tarjan [1987] showed that each scaling phase performs O(nm) saturating pushes, $O(n^2m)$ nonsaturating pushes, and $O(n^2)$ relabels. Using the dynamic tree data structure (Sleator and Tarjan [1983]), the algorithm can be implemented to run in $O(nm \log(n^2/m))$ time per scaling phase and in $O(nm \log(n^2/m) \log(nU))$ total time.

We now describe how to modify the cost-scaling algorithm for the convex cost case. For a given arc flow x', we construct the residual network $G(x')$ in the usual manner. With respect to the flow x', the residual capacity of arc (i, j) is $r_{ij} = M - x'_{ij}$, and the residual capacity of arc (j, i) is $x'_{ij} + M$. The residual network $G(x')$ consists of arcs with positive residual capacity only. We refer to an arc (i, j) in $G(x')$ as a *forward arc* if $(i, j) \in A$, and refer to it as a *backward arc* if $(j, i) \in A$. (Recall from our assumption that either $(i, j) \in A$ or $(j, i) \in A$ but not both.) We denote by A(i) the set of arcs in $G(x')$ emanating from node i.

For a given flow x', we set the cost $c_{ij}(x')$ of a forward arc (i, j) as the right slope of $H_{ij}(\cdot)$ at x'_{ij}. We define the cost $c_{ji}(x')$ of a backward arc (j, i) as the negative of the left slope of $H_{ij}(\cdot)$ at x'_{ij}. Note that if H were a linear function cx, then the cost of arc (i, j) would be c_{ij}, and the cost of the backward arc (j, i) would be $-c_{ij}$, which is the same as the costs of the arcs in $G(x')$ in the linear cost-scaling algorithm. We define the *reduced cost* of a forward arc (i, j) in $G(x')$ as $c^{\pi}_{ij}(x') = c_{ij}(x') - \pi(i) + \pi(j)$, and the reduced cost of a backward arc (j, i) as $c^{\pi}_{ji}(x') = c_{ji}(x') - \pi(j) + \pi(i)$.

We define an arc (i, j) in the residual network as *admissible* as in the linear cost scaling algorithm, that is, $-\varepsilon \le c^{\pi}_{ij}(x') < 0$. For each arc (i, j) in G(x), we define q_{ij}, which denotes the amount of flow that we need to send on arc (i, j) so that both the arcs (i, j) and (j, i) become inadmissible after the push. Also observe that as we send flow on an admissible arc (i, j), the cost of the arc changes as it is a convex function of the flow on the arc. As the cost of the arc changes, its reduced cost also changes accordingly. If arc (i, j) is a admissible arc, then by increasing flow on the arc the cost of the arc (and the reduced cost of the arc) increases (due to the convexity of the arc costs) and eventually it becomes nonnegative, thus making the arc inadmissible. Hence, the number q_{ij} denotes the maximum flow that can be sent on the arc (i, j) so that its reduced cost remains nonpositive. Similarly, when we increase the flow on an admissible arc (i, j), the reduced cost of the backward arc (j, i) decreases, and q_{ij} also denotes the maximum flow that can be sent on the arc (i, j) so that the reduced cost of the arc (j, i) remains nonnegative. We shall show in Lemma 2 that for a forward admissible arc (i, j) in G(x′), q_{ij} is:

$$q_{ij} = \begin{cases} M - x'_{ij} & \text{if } \pi(i) - \pi(j) \geq u_{ij} \\ b_{ij}(\lfloor \pi(i) - \pi(j) \rfloor) - x'_{ij} & \text{if } \pi(i) - \pi(j) < u_{ij} \end{cases}, \tag{12a}$$

and for a backward admissible arc (j, i) in G(x′), q_{ji} is:

$$q_{ji} = \begin{cases} M + x'_{ij} & \text{if } \pi(i) - \pi(j) \leq l_{ij} \\ x'_{ij} - b_{ij}(\lfloor \pi(i) - \pi(j) \rfloor) & \text{if } \pi(i) - \pi(j) > l_{ij} \end{cases}. \tag{12b}$$

Lemma 2. *If arc (i, j) in G(x′) is a forward admissible arc (or a backward admissible arc), and if q_{ij} (or q_{ji}) units of flow are sent on arc (i, j) (or arc (j, i)), then subsequently neither the arc (i, j) nor the arc (j, i) is admissible.*

Proof. Let x″ denote the flow after the push. There are four cases to consider.

Case 1: *Arc (i, j) is a forward arc and $\pi(i) - \pi(j) \geq u_{ij}$.* In this case, as defined by (12), $q_{ij} = M - x'_{ij}$, and the flow x''_{ij} after the push is $x''_{ij} = M$, and thus arc (i, j) $\notin$ G(x″). Moreover, the left slope of $C_{ij}(M)$ (by Lemma 1) equals u_{ij} (or, $c_{ji}(x'') = -u_{ij}$). Thus, $c^{\pi}_{ji}(x'') = c_{ji}(x'') + \pi(i) - \pi(j) = -u_{ij} + \pi(i) - \pi(j) \geq 0$.

Case 2: *Arc (i, j) is a forward arc and $\pi(i) - \pi(j) < u_{ij}$.* In this case, $q_{ij} = b_{ij}(\lfloor\pi(i) - \pi(j)\rfloor) - x'_{ij}$, and so after the push, $x''_{ij} = b_{ij}(\lfloor\pi(i) - \pi(j)\rfloor)$. By Lemma 1, the right slope of $C_{ij}(x''_{ij})$ is at least $\lfloor\pi(i) - \pi(j)\rfloor + 1$ (or, $c_{ij}(x'') \geq \lfloor\pi(i) - \pi(j)\rfloor + 1$) and, therefore,

$$c^{\pi}_{ij}(x'') = c_{ij}(x'') - \pi(i) + \pi(j) \geq (\lfloor\pi(i) - \pi(j)\rfloor + 1) - (\pi(i) - \pi(j)) \geq 0.$$

Also by Lemma 1, the left slope of $C_{ij}(x''_{ij})$ is at most $\lfloor\pi(i) - \pi(j)\rfloor$ (or, $c_{ji}(x'') \geq -\lfloor\pi(i) - \pi(j)\rfloor$) and, therefore,

$$c^{\pi}_{ji}(x'') = c_{ji}(x'') + \pi(i) - \pi(j) \geq -\lfloor\pi(i) - \pi(j)\rfloor + (\pi(i) - \pi(j)) \geq 0.$$

Case 3: *Arc (j, i) is a backward arc, and $\pi(i) - \pi(j) \leq l_{ij}$.* In this case, $x''_{ij} = -M$, and after the push (j, i) $\notin$ G(x″). Moreover, the right slope of $C_{ij}(-M)$ by Lemma 1 equals l_{ij}, and thus $c^{\pi}_{ij}(x'') = c_{ij} - \pi(i) + \pi(j) = l_{ij} - \pi(i) + \pi(j) \geq 0$.

Case 4: *Arc (j, i) is a backward arc, and $\pi(i) - \pi(j) > l_{ij}$.* In this case, $q_{ji} = x'_{ij} - b_{ij}(\lfloor\pi(i) - \pi(j)\rfloor)$, and after the push, the amount of flow in arc (i, j) is $b_{ij}(\lfloor\pi(i) - \pi(j)\rfloor)$, and so the argument given in Case 2 applies.

We also need to prove that $q_{ij} > 0$ (or $q_{ji} > 0$). We will prove this result for the case when arc (i, j) is a forward arc; proof for the case when arc (i, j) is a backward arc is similar. Observe that $c^{\pi}_{ij}(x') < 0$ (because arc (i, j) was an admissible arc before the push) and $c^{\pi}_{ij}(x'') \geq 0$. By the convexity of $C_{ij}(.)$, it follows that $x'_{ij} < x''_{ij}$ and so $q_{ij} > 0$. This completes the proof of the lemma. ♦

Our convex cost-scaling algorithm is almost identical to the linear cost-scaling algorithm. We give an algorithmic description of this algorithm below.

```
algorithm cost scaling
begin
    $\pi := 0$ and $\varepsilon := C$;
    let x be any feasible flow;
    while $\varepsilon \geq 1/n$ do
    begin
        improve-approximation($\varepsilon$, x, $\pi$);
        $\varepsilon := \varepsilon/2$;
    end;
    x is an optimal flow for the dual network flow problem;
end;
```

```
procedure improve-approximation($\varepsilon$, x, $\pi$);
begin
        for every admissible arc (i, j) $\in$ A do send $q_{ij}$ amount of flow on arc (i, j);
        compute node imbalances;
        while the network contains an active node do
        begin
            select an active node i;
            if G(x) contains an admissible arc (i, j) then
                push $\delta := \min\{e(i), q_{ij}\}$ units of flow from node i to node j
           else $\pi(i) := \pi(i) + \varepsilon/2$;
        end;
end;
```

The algorithm proceeds by identifying excess nodes and pushing flows on arcs emanating from these nodes; when there is no such arc on which flow can be pushed, it relabels the node. The convex cost-scaling algorithm is identical to the linear cost-scaling algorithm, it differs only on one count: the convex cost-scaling algorithm treats q_{ij}'s in the same manner as the linear cost scaling algorithm treats the residual capacities r_{ij}'s. In the convex cost-scaling algorithm, we refer to the push on arc (i, j) as *emptying* if $\delta = e(i)$ and *nonemptying* if $\delta < e(i)$ (or, alternatively, $\delta = q_{ij}$). We show next in Lemma 3 that a nonemptying push on an admissible arc (i, j) makes both the arcs (i, j) and (j, i) admissible. (A nonemptying push is similar to the saturating push in the linear cost-scaling algorithm which also makes both the arcs (i, j) and (j, i) inadmissible.) We now bound the running time of the convex cost-scaling algorithm. We accomplish it by bounding the number of relabel operations, the number of nonemptying pushes, and the number of emptying pushes performed in the *improve-approximation* procedure.

Lemma 3. *The improve-approximation procedure always maintains $\varepsilon/2$-optimality of the flow, and at termination yields an $\varepsilon/2$-optimal flow.*

Proof. We prove this lemma by performing induction on the number of pushes and relabels. At the beginning of the procedure, the algorithm sends q_{ij} flow on each admissible arc (i, j) which makes both the arcs (i, j) and (j, i) inadmissible after the push. Since there are no admissible arcs left, $c^{\pi}_{ij} \geq 0$ for each arc (i, j) in the residual network with respect to the current node potentials π, and the current pseudoflow is $\varepsilon/2$-optimal (in fact, it is 0-optimal). Now notice that a node i is relabeled only when $c^{\pi}_{ij} \geq 0$ for each arc (i, j) $\in$ A(i) and this operation increases $\pi(i)$ by $\varepsilon/2$. Hence, $c^{\pi}_{ij} \geq -\varepsilon/2$ for each arc (i, j) $\in$ A(i) and the induction hypothesis holds. Each nonemptying push on arc (i, j) makes both the arcs (i, j) and (j, i) inadmissible and the induction hypothesis still holds. Now consider an emptying push. Recall from our previous discussion that by sending additional flow on the arc (i, j), the reduced cost of the arc (i, j) increases (from a number which is at least $-\varepsilon/2$) and the reduced cost of the arc (j, i) decreases. Also recall from our previous discussion that when we send q_{ij} amount of flow, reduced costs of both the arcs become nonnegative. It follows from these two observations that if we send any flow less than q_{ij} (which an emptying push does), then the reduced cost of arc (i, j) remains between $-\varepsilon/2$ and zero, and the reduced cost of arc (j, i) remains nonnegative. Hence an emptying push preserves the induction hypothesis. The proof of the lemma is now complete.

Lemma 4. *During an execution of the improve-approximation procedure, any node potential increases O(n) times. The total time needed to perform all relabel operations in the improve-approximation procedure is O(nm).*

Proof. Let x be the current $\varepsilon/2$-optimal pseudoflow and x′ be the ε-optimal flow at the end of the previous cost scaling phase. Let π and π' be the node potentials corresponding to the pseudoflow x and the flow x′. Goldberg and Tarjan [1987] established (see, for example, Lemma 10.4 in Ahuja, Magnanti and Orlin [1993]) that during an execution of the *improve-approximation* procedure $\pi(i) - \pi'(i) \leq 3n\varepsilon/2$. The proof of this result uses the fact that the *improve-approximation* procedure maintains $\varepsilon/2$-optimality of the flow at every step. Lemma 3 shows that the *improve-approximation* procedure indeed maintains $\varepsilon/2$-optimality of the flow, hence this result applies to the convex cost scaling algorithm too. Further, since each node potential increases by an increment of $\varepsilon/2$ in the *improve-approximation* procedure, any node potential increases at most 3n = O(n) times. This completes the first part of the lemma. A relabel operation of node i takes O(|A(i)|) time. The total time needed to relabel all nodes is $O(n\sum_{i\in N}|A(i)|) = O(nm)$. This completes the proof of the second part of the lemma. ♦

Lemma 5. *The improve-approximation procedure performs O(nm) nonemptying pushes.*

Proof. We will use the potential function argument on F to bound the number of nonemptying pushes. Let F denote the number of admissible arcs in the residual network G(x′). A nonemptying push on arc (i, j) makes arc (i, j) inadmissible after the push and does not make the arc (j, i) admissible, and so it reduces F by one. An emptying push either decreases F by one or keeps it constant. Each distance increase of node i may create as many as |A(i)| new admissible arcs, and hence may increase F by at most |A(i)|. The total increase in F over all iterations is $O(n\sum_{i\in N}|A(i)|) = O(nm)$. It follows that the number of nonemptying pushes is O(nm). ♦

Lemma 6. *The improve-approximation procedure performs $O(n^2m)$ emptying pushes.*

Proof. Let E denote the set of excess nodes and for an excess node $i \in E$, let g(i) denote the number of nodes in the residual network that are reachable from node i via a directed path of violating admissible arcs. Let $\Phi = \sum_{i\in E} g(i)$. Each emptying push decreases Φ by at least 1. Each nonemptying push increases Φ by at most n. Each distance increase of node i increases Φ by at most n. A standard potential function argument (see, for example, Lemma 10.7 in Ahuja, Magnanti and Orlin [1993]) shows that the number of emptying pushes is $O(n^2m)$. ♦

It can be easily shown that using simple data structures, the improve-approximation procedure for the convex cost-scaling algorithm can be implemented in $O(n^2m)$ time, which is the same time needed to implement the linear cost-scaling algorithm using the same data structures. Goldberg and Tarjan [1987] showed that using the dynamic tree data structure of Sleator and Tarjan [1983], the improve-approximation procedure for the linear cost-scaling algorithm can be implemented in $O(nm \log(n^2/m))$ time. The *improve-approximation* procedure for the convex cost-scaling algorithm is identical to that for the linear cost-scaling algorithm except that in place of the residual capacities r_{ij}'s we use q_{ij}'s and both can be computed in O(1) time for an arc (i, j). Hence all the arguments of the running time analysis of the dynamic tree version of the improve-approximation procedure for the linear cost-scaling algorithm apply for the convex cost case, and we obtain the same running time for the convex cost case. Consequently, we obtain a time bound of $O(nm \log(n^2/m))$ for the *improve-approximation* procedure for the convex cost-scaling algorithm. Each application of the *improve-approximation* procedure reduces ε by a factor of 2. After O(log(nU)) executions of *improve-approximation* procedure, the algorithm obtains an ε-optimal flow with $\varepsilon < 1/n$. It is easy to see that this flow is an optimal flow for the convex cost flow problem (11). This result follows from the fact that the convex cost flow problem (11) can be transformed to a minimum cost flow problem with integer arc costs, and any ε-optimal flow with $\varepsilon < 1/n$ is an optimal flow for the minimum cost flow problem. We have thus shown that the convex cost-scaling algorithm solves (11) in $O(nm \log(n^2/m) \log U)$ time.

The convex cost-scaling algorithm upon termination gives an optimal flow x^* and the optimal node potentials π^*. Both the solutions x^* and π^* may be non-integer. Since the objective function in the convex cost flow problem (11) is piecewise linear, it follows that there always exists an integer optimal node potentials π. To determine those, we construct $G(x^*)$ and solve a shortest path problem to determine shortest path distance $d(i)$ from node 0 to every other node $i \in N$. Since all arc costs in $G(x^*)$ are integer, each $d(i)$ is also integer. Then $\pi(i) = -d(i)$ for each $i \in N$ gives an integer optimal set of node potentials for the problem (11). Now recall that $C_{ij}(x_{ij}) = -H_{ij}(x_{ij})$ for each $(i, j) \in A$. This implies that $\mu(i) = -\pi(i) = d(i)$ for each $i \in N$ gives optimal dual variables for (9) and these $\mu(i)$ together $w_{ij} = \mu(i) - \mu(j)$ for each $(i, j) \in A$ give an optimal solution of the dual network flow problem (3). We summarize our discussion with the following theorem:

Theorem 3. *The convex cost-scaling algorithm correctly solves the dual network flow problem in $O(nm \log(n^2/m) \log(nU))$ time.*

5. Application of the Dual Network Flow Problem

The dual network flow problem and its special cases arise in many application settings. Roundy [1986] formulates a lot-sizing problem in a multi-product, multi-stage, production/inventory system as a dual network flow problem. Boros and Shamir [1991] describe an application of the dual network flow problem in solving a quadratic cost machine scheduling problem. Several multi-facility location problems have constraints of the form (1b) – (1e) (see for example, Ahuja Magnanti and Orlin [1993, Chapter 19]). The convex cost version of these location problems will be dual network flow problems. We describe next in detail five applications of dual network flow problems, some of which we have encountered in our previous research.

Application 1. Dial-A-Ride Transit Problem

Dial-a-ride transit problems are vehicle routing problems with time windows and have been extensively studied in the literature. (See, for example, Desrosiers et al. [1995].) In this problem, customers call a dial-a-ride agency sufficiently in advance (say, one day before) requesting to be carried from specific origins to specific destinations during specified times. The agency dispatches a vehicle to meet several such demands and customers are pooled to reduce the operational costs. A vehicle schedule typically consists of picking up and dropping off of some customers in a specific sequence, and at any point of time several customers can be on board the vehicle. Researchers have developed exact as well as heuristic algorithms for dial-a-ride transit problems. Since exact algorithms can solve only small sized problems, heuristic algorithms have been more extensively studied. Some heuristic algorithms deal separately with the following two functions: routing and scheduling. The routing part determines the route of each vehicle - the order in which specific customers assigned to a vehicle will be picked up and delivered. The scheduling part assigns a time schedule to the route - the times at which the customers will be picked up and delivered. We will show that determining the optimal schedule for a given route can be formulated as the convex cost dual network flow problem.

In this scheduling problem we are given a sequence of stops (on an increasing time scale) 1-2-3- ... -n, where each stop denotes a pickup or a delivery point. Let **S** denote the set of pickup stops, and for each pickup stop $i \in \mathbf{S}$ let d(i) denote the corresponding delivery stop. We assume that the vehicle takes t_j time to go from stop j to stop j+1. Each stop j has a pickup/delivery time window $[l_j, u_j]$ and also has a *desired* pickup/delivery time a_j. We penalize the deviations from the desired pickup and delivery times using a convex function. If the vehicle visits stop j at time μ_j, the penalty is given by a convex function $B_j(\mu_j - a_j)$. We also allow the vehicle to wait between stops, but penalize these waiting times. If w_i denotes the waiting time w_i between stop i and stop i+1, then $w_i = \mu_{i+1} - \mu_i - t_i$. The penalty associated with this waiting time is given by a convex function $F_i(w_i)$. Let τ_i denote the minimum ride time from a pickup stop i to its delivery stop d(i). However, due to the waiting times between stops, the actual ride time, given by $\mu_{d(i)} - \mu_i$, may be larger than τ_i; giving us the excess ride time $e_i = \mu_{d(i)} - \mu_i - \tau_i$. We penalize the excess ride time e_i using the convex function $E_i(e_i)$. Then the scheduling problem is to determine the vehicle schedule so that the total penalty incurred due to the deviations from the desired pickup and delivery time, waiting times, and the excess ride times is minimum. This problem can be formulated as the following optimization problem:

$$\text{Minimize } \sum_{j=1}^{n} [F_i(w_i) + B_i(\mu_i - a_i)] + \sum_{i \in P} E_i(e_i) \qquad (16a)$$

subject to

$w_i = \mu_{i+1} - \mu_i - t_i$ for each i =1, 2, ... , n, (16b)

$e_i = \mu_{d(i)} - \mu_i - \tau_i$ for each $i \in \mathbf{S}$, (16c)

$0 \le w_i \le \bar{w}$ for each i = 1, 2, … , n. (16d)

$0 \le e_i \le \bar{e}_i$ for each $i \in \mathbf{S}$, (16e)

μ_i, w_i, and e_i are all integer, (16f)

where $\bar{w}$ specifies an upper bound on any single waiting time, and $\bar{e}_i$ specifies an upper bound on e_i. This problem can easily be formulated as the dual network flow problem by defining the set P and Q appropriately.

Application 2: Inverse Spanning Tree Problem with Convex Costs

Consider an undirected network G = (N, A) with the node set N and the arc set A. Let n = |N| and m = |A|. We assume that N = {1, 2, ... , n} and A = $\{a_1, a_2, ..., a_m\}$. Let c_j denote the cost of the arc a_j. In the inverse spanning tree problem we are given a spanning tree T^0 of G which may or may not be a minimum spanning tree of G and we wish to perturb the arc cost vector c to d so that T^0 becomes a minimum spanning tree with d as the cost vector and $\sum_{j=1}^{n} F_j(d_j - c_j)$ is minimum, where each $F_j(d_j - c_j)$ is a convex function of d_j. Sokkalingam, Ahuja and Orlin [2000], and Ahuja and Orlin [2000] have studied special cases of the inverse spanning tree problem with cost functions such as $\sum_{j=1}^{n} | d_j - c_j |$ and $\max\{|d_j - c_j| : 1 \le j \le m\}$. Hochbaum [2001b] studied the problem also for general convex functions.

We assume without any loss of generality that $T^0 = \{a_1, a_2, ... , a_{n-1}\}$. We refer to the arcs in T^0 as *tree arcs* and the arcs not in T^0 as *nontree arcs*. In the given spanning tree T^0, there is a unique path between any two nodes; we denote by $W[a_j]$ the set of tree arcs contained between the two endpoints of the arc a_j. It is well known (see, for example, Ahuja, Magnanti and Orlin [1993]) that T^0 is a minimum spanning tree with respect to the arc cost vector d if and only if

$d_i \le d_j$ for each $a_i \in W[a_j]$ and for each j = n, n+1, ..., m. (17)

We can convert the inequalities in (17) into equations by introducing nonnegative slack variables w_{ij}'s. Let $P = \{1, 2, \ldots, m\}$ and $Q = \{(i, j) : a_i \in T^0 \text{ and } a_j \in W[a_i]\}$. Then, in this notation, the inverse spanning tree problem can be formulated as the following optimization problem:

$$\text{Minimize } \sum_{i \in P} F_i(d_i - c_i) \tag{18a}$$

subject to

$$d_i - d_j = w_{ij} \qquad \text{for each } (i, j) \in Q, \tag{18b}$$

$$w_{ij} \geq 0 \qquad \text{for each } (i, j) \in Q. \tag{18c}$$

This problem is an instance of the dual network flow problem.

Application 3: Time-Cost Tradeoff Problem in Project Scheduling

Project scheduling is regularly carried out in numerous industries. We will show that the *time-cost tradeoff problem*, an important problem in project scheduling, can be formulated as a dual network flow problem. We refer the reader to the book by Elmaghraby [1978] for a comprehensive discussion of the applications of network models in project scheduling.

We can envision a project as a directed graph $G = (N, A)$ where the arc set A represents the jobs of the project and the node set N represents *events*, denoting the beginning and ending of jobs. Different jobs in the network have precedence relations. We require that all jobs directed into any node must be completed before any job directed out of the node begins. We let node s designate the beginning of the project and node T designate the ending of the project.

Normally, in project scheduling we assume that the job completion time is fixed; however, we here consider a more general settings where job completion times are variable. Let t_{ij} denote the time it takes to complete the job (i, j). We allow t_{ij} to vary in the range $[\alpha_{ij}, \beta_{ij}]$ and associate a convex cost function $F_{ij}(t_{ij})$ which captures the cost of completing the job for different completion times in the range $[\alpha_{ij}, \beta_{ij}]$. (These are often treated in the literature as "crash times.") We also have another set of decision variables υ_i's denoting *event times*; the variable υ_i denotes the time when jobs emanating from node i can begin. The time-cost tradeoff problem is to determine the minimum cost of completing the project in a specified time period T. This problem can be mathematically stated as follows:

$$\text{Minimize } \sum_{(i,j)\in A} F_{ij}(t_{ij}) \tag{19a}$$

subject to

$$\upsilon_t - \upsilon_s \leq T \tag{19b}$$

$$\upsilon_j - \upsilon_i \geq t_{ij} \qquad \text{for all } (i, j) \in A, \tag{19c}$$

$$\alpha_{ij} \leq t_{ij} \leq \beta_{ij} \qquad \text{for all } (i, j) \in A. \tag{19d}$$

Observe that the constraints (19b) capture the fact that the project must be completed within the time period T, and the constraints (19c) imply that for every arc $(i, j) \in A$, event j must occur at least w_{ij} time units later than the occurrence of event i. The formulation (19) is an instance of the dual network flow problem.

Erenguc et al. [2001] describe a generalization of the time-cost tradeoff problem in project scheduling where tasks have multiple crashable modes. They show that this problem can be formulated as the dual network flow problem.

Application 4: Just-In-Time Scheduling in Project Management

Just-in-time is a management philosophy that has become quite popular in recent years. It attempts to eliminate waste by reducing slack times. We describe here an application of just-in-time scheduling applied to project management. This application has been adapted by us from Levner and Nemirovsky [1991]. As in the previous application, we denote a project by a directed graph G = (N, A), where set A denotes *jobs*, and the node set N denotes *events*. The network G also captures precedence relations among the arcs. We assume in this application that the completion times of all jobs are fixed. Let t_{ij} denote the time it takes to complete job (i, j). (Notice that t_{ij} is not a decision variable in this problem.) Let υ_i denote the time for event i.

Consider the feasible event times υ_i's that is, satisfying $\upsilon_j - \upsilon_i \geq t_{ij}$ for all $(i, j) \in A$. With respect to these event times, a job (i, j) will be completed at time $\upsilon_i + t_{ij}$ but the jobs emanating from node j will start at time υ_j. Let $w_{ij} = \upsilon_j - \upsilon_i - t_{ij}$ denote the *slack time*, and let $F_{ij}(w_{ij})$ denote its associated penalty cost. This penalty cost may capture the lost opportunity cost of the capital tied or some other factors (such as, perishability or deterioration in quality) which make slack times undesirable. There may also be some upper bounds β on slack times. We may assume without loss of generality that the lower bound on slack time is 0, since any other lower bound would be incorporated into the times to complete a task. The just-in-time project scheduling problem is to obtain event times υ_i's so that the project is completed within the specified time period T and the penalty cost associated with job slacks is minimum. This problem can be mathematically modeled as follows:

$$\text{Minimize } \textstyle\sum_{(i,j)\in A} F_{ij}(w_{ij}) \tag{20a}$$

subject to

$$\upsilon_t - \upsilon_s \le T \tag{20b}$$

$$\upsilon_j - \upsilon_i - w_{ij} = t_{ij} \qquad \text{for all } (i, j) \in A, \tag{20c}$$

$$0 \le w_{ij} \le \beta_{ij} \qquad \text{for all } (i, j) \in A. \tag{20d}$$

Application 5. Isotonic Regression Problem

The *isotonic regression problem* can be defined as follows. Given a set $A = \{a_1, a_2, \dots, a_n\} \in R^n$, find $X = \{x_1, x_2, \dots, x_n\} \in R^n$ so as to

$$\text{Minimize } \sum_{j=1}^{n} B_j(x_j - a_j) \tag{21a}$$

subject to

$$x_j \le x_{j+1} \qquad \text{for all } j = 1, 2, \dots, n\text{-}1, \tag{21b}$$

$$l_j \le x_j \le u_j \qquad \text{for all } j = 1, 2, \dots, n\text{-}1, \tag{21c}$$

$$x_j \text{ integer} \qquad \text{for all } j = 1, 2, \dots, n\text{-}1. \tag{21d}$$

where $B_j(x_j - a_j)$ is a convex function for every j, $1 \le j \le n$. The isotonic regression problem arises in statistics, production planning, and inventory control (see for example, Barlow et al. [1972] and Robertson et al. [1988]). As an illustration of the isotonic regression, consider a full tank where fuel is being consumed at a slow rate and measurements of the fuel take are being taken at different points in time. Suppose these measurements are $a_1, a_2, .., a_n$. Due to the errors in the measurements, these numbers may not be in the nonincreasing order despite the fact that the true amounts of fuel remaining in the tank are nonincreasing. However, we need to determine these measurements as accurately as possible. One possible way to accomplish this could be to perturb these numbers to $x_1 \geq x_2 \geq \ldots \geq x_n$ so that the cost of perturbation given by $\sum_{j=1}^{n} B_j(x_j - a_j)$ is minimum, where each $B_j(x_j - a_j)$ is a convex function. We can transform this problem to the isotonic regression problem by replacing x_j's by their negatives.

If we define P = {1, 2, … , n} and Q = {(j, j+1) : j = 1, 2, … , n-1} and require that x_j must be integer, then the isotonic regression problem can be cast as a dual network flow problem. However, it is a very special case of the dual network flow problem and more efficient algorithms can be developed to solve it compared to the dual network flow problem. Ahuja and Orlin [2001] recently developed an O(n log U) algorithm to solve the isotonic regression problem. Hochbaum and Queyranne [2000] described an algorithm for the isotonic regression of complexity O(n log n+ n log U) where the second term in the complexity expression is the work required to find the integer minima of functions in the objective. These functions are in general n convex functions in which case the complexity is as stated. In some cases the functions are simpler and the second term can be implemented more efficiently.

ACKNOWLEDGMENTS

We thank the referees for their insightful comments and suggestions which led to an improved presentation of results. The first author gratefully acknowledges the support of NSF Grant DMI-9900087 and DMI-0085682. The research of the second author was supported by NSF grants DMI-0085690 and DMI-0084857. The research of the third author was supported by the NSF Grant DMI-9820998 and the Office of Naval Research Grant ONR N00014-98-1-0317.

REFERENCES

Ahuja, R. K., D.S. Hochbaum, and J.B. Orlin. 1999. Solving the convex cost integer dual network flow problem. *Proceedings of IPCO'99 (Integer Programming and Combinatorial Optimization)*, Edited by G. Cornuejols, R. E. Burkard, and G. J. Woeginger*, Lecture Notes in Computer Science* **1610**, 31-44.

Ahuja, R.K., D.S. Hochbaum, and J.B. Orlin. 2000. A cut based algorithm for the nonlinear dual of the minimum cost network flow problem. Technical Report, Dept. of Industrial Engineering and Operations Research, University of California, Berkeley, CA.

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

Ahuja, R.K., and J.B. Orlin. 2000. A faster algorithm for the inverse spanning tree problem. *Journal of Algorithms* **34**, 177-193.

Ahuja, R.K., and J.B. Orlin. 2001. A fast scaling algorithm for minimizing separable convex functions subject to chain constraints. *Operations Research* 49, 784-789.

Barlow, R.E., D.J. Bartholomew, D.J. Bremner, and H.D. Brunk. 1972. *Statistical Inference under Order Restrictions.* Wiley, New York.

Boros, E., and R. Shamir. 1991. Convex separable minimization over partial order constraints. Report No. RRR 27-91, RUTCOR, Rutgers University, New Brunswick, NJ.

Desrosiers, J., Y. Dumas, M.M. Solomon, and F. Soumis. 1995. Time constrained routing and scheduling. *Handbooks in Operations Research and Management Science, Volume 8: Network Routing*, North Holland, Amsterdam.

Elmaghraby, S.E. 1978. *Activity Networks: Project Planning and Control by Network Models*. Wiley-Interscience, New York.

Erenguc, S.S., T. Ahn, and D.G. Conway. 2001. The resource constrained project scheduling problem with multiple crashable modes: An exact solution method. *Naval Research Logistics* **48**, 107-127.

Goldberg, A.V., and R.E. Tarjan. 1986. A new approach to the maximum flow problem. *Proceedings of the 18th ACM Symposium on the Theory of Computing*, 136-146. Full paper in *Journal of ACM* **35**(1988), 921-940.

Goldberg, A.V., and R.E. Tarjan. 1987. Solving minimum cost flow problem by successive approximation. *Proceedings of the 19th ACM Symposium on the Theory of Computing,* 7-18. Full paper in *Mathematics of Operations Research* **15**(1990), 430-466.

Goldberg, A.V., and R.E. Tarjan. 1988. Finding minimum-cost circulations by cancelling negative cycles. *Proceedings of the 20th ACM Symposium on the Theory of Computing,* 388-397. Full paper in *Journal of ACM* **36**(1989), 873-886.

Hochbaum, D.S. 2001a. An efficient algorithm for image segmentation, Markov random fields and related problems. *Journal of ACM* **48**, 686-701.

Hochbaum, D.S. 2001b. Efficient algorithms for the inverse spanning tree problem. Technical Report, Dept. of IE and OR, University of California, Berkeley, CA.

Hochbaum, D.S., and M. Queyranne. 2000. Minimizing a convex cost closure set. Technical Report, *Lecture Notes in Computer Science 1879. Mike Paterson (ed.) Eighth Annual European Symposium on Algorithms* - ESA 2000 Saarbrucken, Germany, September 2000, pp. 256-267. To appear in *SIAM Journal on Discrete Mathematics*.

Hochbaum, D.S., and J.G. Shanthikumar. 1990. Convex separable optimization is not much harder than linear optimization. *Journal of ACM* **37**, 843-862.

Karzanov, A.V., and S.T. McCormick. 1997. Poloynomial methods for separable convex optimization in unimodular linear spaces with applications. *SIAM Journal on Computing* **4**, 1245-1275.

Levner, E.V., and A.S. Nemirovsky. 1991. A network flow algorithm for just-in-time project scheduling. Memorandum COSOR 91-21, Department of Mathematics and Computing Science, Eindhoven University of Technology, Eindhoven, The Netherlands.

Murty, K.G. 1985. *Linear and Combinatorial Programming*, R. E. Krieger.

Robertson, T., F.T. Wright, and R.L. Dykstra. 1988. *Order Restricted Statistical Inference*. John Wiley & Sons, New York.

Rockafellar, R.T. 1984. *Network Flows and Monotropic Optimization*. Wiley-Interscience, New York.

Roundy, R. 1986. A 98% effective lot-sizing rule for a multi-product, multi-stage, production-/inventory system. *Mathematics of Operations Research* **11**, 699-727.

Sleator, D.D., and R.E. Tarjan. 1983. A data structure for dynamic trees. *Journal of Computer and System Sciences* **24**, 362-391.

Sokkalingam, P.T., R.K. Ahuja, and J.B. Orlin. 1999. Solving inverse spanning tree problems through network flow techniques. *Operations Research* **47***,* 291-300.

Tarjan, R.E. 1998. Personal communications.