April 1996 (revised **October 1996**)

$\mathcal{E}$-RELAXATION AND AUCTION METHODS FOR SEPARABLE CONVEX COST NETWORK FLOW PROBLEMS[1]

by

Dimitri P. Bertsekas,[2] Lazaros C. Polymenakos,[3] and Paul Tseng[4]

Abstract

We consider a generic auction method for the solution of the single commodity, separable convex cost network flow problem. This method provides a unifying framework for the ϵ-relaxation method and the auction/sequential shortest path algorithm and, as a consequence, we develop a unified complexity analysis for the two methods. We also present computational results showing that these methods are much faster than earlier relaxation methods, particularly for ill-conditioned problems.

[1] Research supported by NSF under Grant CCR-9103804 and Grant 9300494-DMI.

[2] Department of Electrical Engineering and Computer Science, M.I.T., Room 35-210, Cambridge, Massachusetts 02139. Email address: dimitrib@mit.edu

[3] IBM T. J. Watson Research Center, Room 23-116C, Yorktown Heights, New York 10598. Email address: lcp@watson.ibm.com

[4] Department of Mathematics, University of Washington, Seattle, Washington 98195. Email address: tseng@math.washington.edu

1. INTRODUCTION

We consider a directed graph with node set $\mathcal{N} = \{1, \ldots, N\}$ and arc set $\mathcal{A} \subset \mathcal{N} \times \mathcal{N}$, where N denotes the number of nodes and A denotes the number of arcs. (The implicit assumption that there exists at most one arc in each direction between any pair of nodes is made for notational convenience and can be dispensed with.) We are given, for each node $i \in \mathcal{N}$, a scalar s_i (the *supply* of i) and, for each arc $(i,j) \in \mathcal{A}$, a convex, closed, proper function $f_{ij} : \Re \to \Re \cup \{\infty\}$ (the *cost function* of (i,j)), i.e., f_{ij} is extended real-valued, lower semicontinuous, not identically taking the value ∞ [Roc70]. The convex cost network flow problem with separable cost function is

$$\begin{aligned} &\text{minimize} \quad f(x) = \sum_{(i,j)\in\mathcal{A}} f_{ij}(x_{ij}) && \text{(P)} \\ &\text{subject to} \quad \sum_{\{j \mid (i,j)\in\mathcal{A}\}} x_{ij} - \sum_{\{j \mid (j,i)\in\mathcal{A}\}} x_{ji} = s_i, \qquad \forall\, i \in \mathcal{N}, && (1) \end{aligned}$$

where the real variable x_{ij} is referred to as the *flow* of the arc (i,j) and the vector $x = \{x_{ij} \mid (i,j) \in \mathcal{A}\}$ is referred to as the *flow vector*. We refer to problem (P) as the *primal* problem. A flow vector x with $f_{ij}(x_{ij}) < \infty$ for all $(i,j) \in \mathcal{A}$, which satisfies the conservation-of-flow constraint (1) is called *feasible*. For a given flow vector x, the *surplus* of node i is defined as the difference between the supply s_i and the net outflow from i:

$$g_i = s_i + \sum_{\{j \mid (j,i)\in\mathcal{A}\}} x_{ji} - \sum_{\{j \mid (i,j)\in\mathcal{A}\}} x_{ij}. \tag{2}$$

We will assume that *there exists at least one feasible flow vector x such that*

$$f_{ij}^-(x_{ij}) < \infty \ \text{ and } \ f_{ij}^+(x_{ij}) > -\infty, \qquad \forall\, (i,j) \in \mathcal{A}, \tag{3}$$

where $f_{ij}^-(x_{ij})$ and $f_{ij}^+(x_{ij})$ denote the left and right directional derivative of f_{ij} at x_{ij} [Roc84, p. 329].

There is a well-known duality framework for this problem, primarily developed by Rockafellar [Roc70], and discussed in several texts; see e.g. [Roc84], [BeT89]. This framework involves a Lagrange multiplier p_i for the ith conservation-of-flow constraint (1). We refer to p_i as the *price* of node i, and to the vector $p = \{p_i \mid i \in \mathcal{N}\}$ as the *price vector*. The *dual* problem is

$$\begin{aligned} &\text{minimize} \quad q(p) && \text{(D)} \\ &\text{subject to} \quad \text{no constraint on } p, \end{aligned}$$

where the dual functional q is given by

$$q(p) = \sum_{(i,j)\in\mathcal{A}} q_{ij}(p_i - p_j) - \sum_{i\in\mathcal{N}} s_i p_i,$$

and q_{ij} is related to f_{ij} by the conjugacy relation

$$q_{ij}(t_{ij}) = \sup_{x_{ij} \in \Re} \{x_{ij}t_{ij} - f_{ij}(x_{ij})\}.$$

We will assume throughout that f_{ij} *is such that* q_{ij} *is real-valued for all* $(i,j) \in \mathcal{A}$. This is true, for example, if each function f_{ij} has finite value inside some compact interval and takes the value ∞ outside of the interval. Of particular importance is the *linear cost* case in which f_{ij} is linear inside the interval [FoF62], [Ber91], [Roc84].

It is known (see [Roc84, p. 360]) that, under our assumptions, both the primal problem (P) and the dual problem (D) have optimal solutions and their optimal costs are the negatives of each other. Moreover, a necessary and sufficient condition for a flow-price vector pair (x,p) to be primal and dual optimal is that x is feasible and (x,p) satisfies the *complementary slackness* (CS for short) conditions:

$$f_{ij}^-(x_{ij}) \le p_i - p_j \le f_{ij}^+(x_{ij}), \qquad \forall\ (i,j) \in \mathcal{A}.$$

We will be interested in the following relaxed version of the CS conditions, first introduced in [BHT87]: We say that a flow-price vector pair (x,p) satisfies the *ϵ-complementary slackness* (ϵ-CS for short) conditions, where ϵ is any positive scalar, if

$$f_{ij}(x_{ij}) < \infty, \quad \text{and} \quad f_{ij}^-(x_{ij}) - \epsilon \le p_i - p_j \le f_{ij}^+(x_{ij}) + \epsilon, \qquad \forall\ (i,j) \in \mathcal{A}. \tag{4}$$

There are three classes of methods for solving the problem (P) and its dual (D) for the linear cost case: primal, dual, and auction methods. The primal and dual methods iteratively improve the primal or the dual cost function. The auction approach, which may not improve the primal or the dual cost at any iteration, was introduced in the original proposal of the auction algorithm for the assignment problem [Ber79], and the subsequent ϵ-relaxation method [Ber86a], [Ber86b]. These methods iteratively adjust x and p, one component at a time, so as to drive the node surpluses to zero while maintaining ϵ-CS at all iterations. They have an excellent worst-case (computational) complexity, when properly implemented, as shown in [Gol87] (see also [BeE88], [BeT89], [GoT90]). Their practical performance is also very good and they are well suited for parallel implementation (see [BCEZ95], [LiZ94], [NiZ93]). Recently, the ϵ-relaxation method was extended to the general problem (P) and its dual (D) by the authors [BPT95] (also see the Ph.D. thesis of the second author [Pol95]) and, independently, by De Leone et al. [DMZ95]. These studies report favorable computational experiences with the method, and references [BPT95] and [Pol95] also show that the method has a good worst-case complexity.

In this paper, we consider a generic auction method for solving (P) and (D), whereby x and p are alternately adjusted so as to drive the node surpluses to zero while maintaining ϵ-CS at all iterations. The only additional requirements are that nodes with nonnegative surplus continue to have nonnegative surplus and that price changes are effected by increasing the price of a node with positive surplus by the maximum amount possible. We then consider two important special cases of this generic method. The first is the ϵ-relaxation method; the second is an extension of the auction/sequential-shortest-path algorithm for the linear cost case [Ber92] to the general convex cost case of (P) and (D). The second method was proposed in the Ph.D. thesis of the second author [Pol95] but otherwise is unpublished. It differs from the first in that, instead of moving flow from nodes with positive surplus to any other nodes along push-list arcs, it moves flow from nodes with positive surplus to nodes with negative surplus along paths comprising push-list arcs. We analyze the (computational) complexity of these two methods and report some favorable computational experience with them. In particular, our test results show that, on problems where some (possibly all) arcs have strictly convex cost, the new methods outperform, often by an impressive margin, earlier relaxation methods. Furthermore, our methods seem to be minimally affected by ill-conditioning in the dual problem. We do not know of any other method for which this is true. We note that there are available other approaches for dealing with separable convex cost network flow problems. These include reducing the problem to an essentially linear cost problem by piecewise linearization of the arc cost functions [Mey79], [KaM84], [Roc84]; primal cost improvement [Wei74], [Roc84], [KaM93]; dual cost improvement based on ϵ-subgradient [Roc84] or ϵ-CS [BHT87]. However, these other approaches tend to be more complicated and their complexities do not match those obtained from the auction approach. The approach of using differentiable unconstrained optimization methods on the dual problem [BHT87], [Ven91], [HaH93], [Hag92], [BeE87], [TBT90], though popular, applies primarily to problems with strictly convex arc cost functions.

This paper is organized as follows. In Section 2 we present the generic auction method and analyze its termination property. In Section 3, we consider the first special case of the generic method, the ϵ-relaxation method, and we analyze its complexity using the results of Section 2. In Section 4 we consider the second special case of the generic method, the auction/sequential-shortest-path algorithm, and we analyze its complexity also using the results of Section 2. Finally, in Section 5, we report some of our computational experience with the methods of Sections 3 and 4 on some convex quadratic cost problems. A brief word about notation: By a *path* in $(\mathcal{N}, \mathcal{A})$, we mean a sequence of nodes $(n_1, n_2, \ldots, n_k)$ in $\mathcal{N}$ and a corresponding sequence of $k-1$ arcs in $\mathcal{A}$ $(k \in \{1, 2, ...\})$ such that the ith arc in the sequence is either (n_i, n_{i+1}) (in which case it is called a *forward* arc) or (n_{i+1}, n_i) (in which case it is called a *reverse* arc). A path with $n_1 = n_k$ is called a *cycle*. A path having no repeated nodes is called *simple*.

2. A GENERIC AUCTION METHOD

Intuitively, a feasible flow vector x and a price vector p that together satisfy the ϵ-CS conditions are approximately optimal for the primal problem (P) and the dual problem (D), respectively. This intuition was verified in a result of [BHT87] which is restated in Prop. 4 to follow. Thus, we may consider finding, for a given $\epsilon > 0$, a feasible flow-price vector pair that satisfies the ϵ-CS conditions and, by making ϵ small enough, we can get as close to optimality as desired. In this section we present a generic method, based on the auction approach, that finds such a pair. We also give a partial complexity analysis for this generic method. In Sections 3 and 4, we will refine our analysis for two special cases of this method, the ϵ-relaxation method of [BPT95] and a certain auction/sequential-shortest-path algorithm.

For a fixed $\epsilon > 0$ and $\beta \in (0, 1)$, and a given flow-price vector pair (x, p) satisfying ϵ-CS, an iteration of the generic auction method updates (x, p) as follows:

An Iteration of the Generic Auction Method

> If there is no node with positive surplus, terminate the method. Otherwise, perform one of the following two operations:
>
> (a) *[Flow adjustment]* Adjust the flow vector x in such a way that ϵ-CS is maintained and all nodes with nonnegative surplus continue to have nonnegative surplus. (Here p is unchanged.)
>
> (b) *[Price rise on a node]* Increase the price p_i of some node i with positive surplus by the maximum amount that maintains ϵ-CS. (Here x and all other components of p are unchanged.)

(Notice that the method either adjusts x with p fixed or adjusts p with x fixed. We can more generally consider adjusting x and p simultaneously and/or adjusting more than one prices at a time, as is done for example in [BHT87]. The analysis below extends accordingly.) Upon termination of the generic auction method, the flow-price vector pair (x, p) satisfies ϵ-CS and all nodes have non-positive surplus. Since we assumed there exists at least one feasible flow vector so that $\sum_{i \in \mathcal{N}} s_i = 0$, it is well known and not difficult to show (by summing Eq. (2) over all nodes i) that all nodes must have zero surplus, i.e., x is feasible. Thus, the validity of the method rests on whether it terminates finitely. In the following proposition, we show that the total number of price rises is finite under a suitable assumption. The proof of this result is identical to that given in [BPT95, Prop. 3] for the ϵ-relaxation method, except $1/2$ is replaced throughout by β. The proof is included for completeness.

Proposition 1: Let K be any nonnegative scalar such that the initial price vector p^0 for the generic auction method (with parameters $\epsilon > 0$ and $\beta \in (0, 1)$) satisfies $K\epsilon$-CS together with some feasible flow vector x^0. Also, assume that each price rise on a node increases the price of that node by at least $\beta\epsilon$, for some fixed $\beta \in (0, 1)$. Then, the method performs at most $(K + 1)(N - 1)/\beta$ price rises on each node.

Proof: Consider the pair (x, p) at the beginning of an iteration of the generic method. Since the surplus vector $g = (g_1, \ldots, g_N)$ is not zero, and the flow vector x^0 is feasible, we conclude that for each node s with $g_s > 0$ there exists a node t with $g_t < 0$ and a simple path H from t to s such that:

$$x_{ij} > x_{ij}^0, \qquad \forall\ (i, j) \in H^+, \tag{5}$$

$$x_{ij} < x_{ij}^0, \qquad \forall\ (i, j) \in H^-, \tag{6}$$

where H^+ is the set of forward arcs of H and H^- is the set of backward arcs of H. [This can be seen from the Conformal Realization theorem ([Roc84] or [Ber91]) as follows. For the flow vector $x - x^0$, the net outflow from node t is $-g_t > 0$ and the net outflow from node s is $-g_s < 0$ (here we ignore the flow supplies), so, by the Conformal Realization Theorem, there is a simple path H from t to s that conforms to the flow $x - x^0$, that is, $x_{ij} - x^0_{ij} > 0$ for all $(i,j) \in H^+$ and $x_{ij} - x^0_{ij} < 0$ for all $(i,j) \in H^-$. Eqs. (5) and (6) then follow.]

From Eqs. (5) and (6), and the convexity of the functions f_{ij} for all $(i,j) \in \mathcal{A}$, we have

$$f^-_{ij}(x_{ij}) \geq f^+_{ij}(x^0_{ij}), \qquad \forall\ (i,j) \in H^+, \tag{7}$$

$$f^+_{ij}(x_{ij}) \leq f^-_{ij}(x^0_{ij}), \qquad \forall\ (i,j) \in H^-. \tag{8}$$

Since the pair (x,p) satisfies ϵ-CS, we also have that

$$p_i - p_j \in [f^-_{ij}(x_{ij}) - \epsilon, f^+_{ij}(x_{ij}) + \epsilon], \qquad \forall\ (i,j) \in \mathcal{A}. \tag{9}$$

Similarly, since the pair (x^0, p^0) satisfies $K\epsilon$-CS, we have

$$p^0_i - p^0_j \in [f^-_{ij}(x^0_{ij}) - K\epsilon, f^+_{ij}(x^0_{ij}) + K\epsilon], \qquad \forall\ (i,j) \in \mathcal{A}. \tag{10}$$

Combining Eqs. (7)-(10), we obtain

$$p_i - p_j \geq p^0_i - p^0_j - (K+1)\epsilon, \qquad \forall\ (i,j) \in H^+,$$

$$p_i - p_j \leq p^0_i - p^0_j + (K+1)\epsilon, \qquad \forall\ (i,j) \in H^-.$$

Applying the above inequalities for all arcs of the path H, we get

$$p_t - p_s \geq p^0_t - p^0_s - (K+1)|H|\epsilon, \tag{11}$$

where $|H|$ denotes the number of arcs of the path H. Since only nodes with positive surplus can change their prices and nodes with nonnegative surplus continue to have nonnegative surplus, it follows that if a node has negative surplus at some time, then its price is unchanged from the beginning of the method until that time. Thus $p_t = p^0_t$. Since the path is simple, we also have that $|H| \leq N - 1$. Therefore, Eq. (11) yields

$$p_s - p^0_s \leq (K+1)|H|\epsilon \leq (K+1)(N-1)\epsilon. \tag{12}$$

Since only nodes with positive surplus can increase their prices and, by assumption, each price rise increment is at least $\beta\epsilon$, we conclude from Eq. (12) that the total number of price rises that can be performed for node s is at most $(K+1)(N-1)/\beta$. **Q.E.D.**

The preceding proposition shows that the bound on the number of price rises is independent of the cost functions, but depends only on

$$K^0 = \min\{K \in [0,\infty) \mid (x^0, p^0) \text{ satisfies } K\epsilon\text{-CS for some feasible flow vector } x^0\ \},$$

which is the minimum multiplicity of ϵ by which CS is violated by the initial price vector together with some feasible flow vector. Note that K^0 is well defined for any p^0 because, for all K sufficiently large, $K\epsilon$-CS is satisfied by p^0 and the feasible flow vector x satisfying Eq. (3).

To ensure that the number of flow adjustments between successive price rises is finite and that each price rise is at least $\beta\epsilon$, we need to further specify how the price rises and flow adjustments should be effected. In the remainder of this section, we introduce the key mechanisms for achieving this. For any $\epsilon > 0$, any $\beta \in (0,1)$, and any flow-price vector pair (x,p) satisfying ϵ-CS, we define for each node $i \in \mathcal{N}$ its *push list* as the union of the following two sets of arcs

$$L^+(i) = \left\{(i,j) \mid \ (1-\beta)\epsilon < p_i - p_j - f_{ij}^+(x_{ij}) \leq \ \epsilon\right\}, \tag{13a}$$

$$L^-(i) = \left\{(j,i) \mid -(1-\beta)\epsilon > p_j - p_i - f_{ji}^-(x_{ji}) \geq -\epsilon\right\}. \tag{13b}$$

Our definition of the push list is a direct extension of that used in [BPT95] for the case $\beta = 1/2$.

For each arc (i,j) (respectively, (j,i)) in the push list of i, the supremum of δ for which

$$p_i - p_j \geq f_{ij}^+(x_{ij} + \delta)$$

(respectively, $p_j - p_i \leq f_{ji}^-(x_{ji} - \delta)$) is called the *flow margin* of the arc. An important fact, observed in [BPT95, Prop. 1] for the case $\beta = 1/2$, is that the flow margin of these arcs are always positive.

Proposition 2: All arcs in the push list of a node have positive flow margins.

Proof: Assume that for an arc $(i,j) \in \mathcal{A}$ we have

$$p_i - p_j < f_{ij}^+(x_{ij} + \delta), \quad \forall\ \delta > 0.$$

Since the function f_{ij}^+ is right continuous, this yields

$$p_i - p_j \leq \lim_{\delta \downarrow 0} f_{ij}^+(x_{ij} + \delta) = f_{ij}^+(x_{ij}),$$

and thus, based on the definition of Eq. (13a), (i,j) cannot be in the push list of node i. A similar argument shows that an arc $(j,i) \in \mathcal{A}$ such that

$$p_j - p_i > f_{ji}^-(x_{ji} - \delta), \quad \forall\ \delta > 0,$$

cannot be in the push list of node i. **Q.E.D.**

The way we will make flow adjustments is to decrease the surplus of a node with positive surplus by increasing/decreasing flow on push-list arcs. (This can be done either one arc at a time, as in the case of the ϵ-relaxation method of Section 3, or one path of arcs at a time, as in the case of the auction/sequential-shortest-path algorithm of Section 4.) When the push list of the node is empty, we perform a price rise on the node. An important fact, observed in [BPT95, Prop. 2] for the case $\beta = 1/2$, is that the price rise increment for a node with empty push list is at least $\beta\epsilon$.

Proposition 3: If we perform a price rise on a node whose push list is empty, then the price of that node increases by at least $\beta\epsilon$.

Proof: If the push list of a node i is empty, then for every arc $(i,j) \in \mathcal{A}$ we have $p_i - p_j - f_{ij}^+(x_{ij}) \leq (1-\beta)\epsilon$, and for every arc $(j,i) \in \mathcal{A}$ we have $p_j - p_i - f_{ji}^-(x_{ji}) \geq -(1-\beta)\epsilon$. This implies that the following numbers:

$$p_j - p_i + f_{ij}^+(x_{ij}) + \epsilon, \qquad \forall\ (i,j) \in \mathcal{A},$$

$$p_j - p_i - f_{ji}^-(x_{ji}) + \epsilon, \qquad \forall\ (j,i) \in \mathcal{A},$$

are all greater than or equal to $\beta\epsilon$. Since a price rise on i increases p_i by the increment that is the minimum of all these numbers, the result follows. **Q.E.D.**

Props. 2 and 3 may be interpreted graphically in terms of the characteristic curve:

$$\Gamma_{ij} = \left\{ (x_{ij}, t_{ij}) \in \Re^2 \mid f_{ij}^-(x_{ij}) \leq t_{ij} \leq f_{ij}^+(x_{ij}) \right\}.$$

Then, (i,j) being in the push list of i (respectively, j) corresponds to $(x_{ij}, p_i - p_j)$ belonging to the "strip" at height between $(1-\beta)\epsilon$ and ϵ above (respectively, below) Γ_{ij}. Figure 1 illustrates when an arc (i,j) is in the push list of i and when it is in the push list of j.

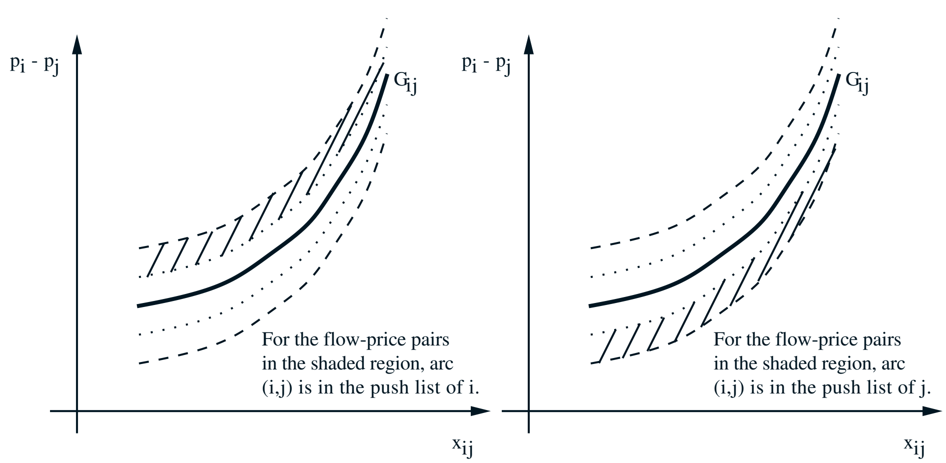

Figure 1: A visualization of the conditions satisfied by a push-list arc. The shaded area represents flow-price differential pairs corresponding to a push-list arc (i,j).

Since f_{ij} is convex so that Γ_{ij} is a monotone curve, it is readily seen that, if (i,j) is in the push list of i (respectively, j), then x_{ij} may be increased (respectively, decreased) by a positive amount before $(x_{ij}, p_i - p_j)$ reaches Γ_{ij}. The flow margin of an arc (i,j) for the case $\beta = 1/2$ is illustrated in Fig. 2. Similarly, if (i,j) is

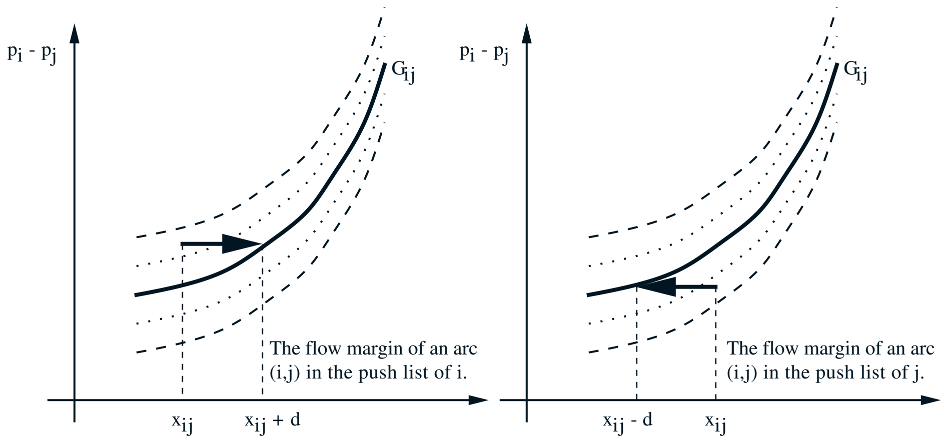

Figure 2: The flow margin δ of a push-list arc (i, j).

not in the push list of i (respectively, j), then $p_i - p_j$ may be increased (respectively, decreased) by at least $\beta\epsilon$ before $(x_{ij}, p_i - p_j)$ exits from the strip of height up to ϵ above and below Γ_{ij}.

Lastly, for any $\epsilon > 0$, any $\beta \in (0, 1)$, and any flow-price vector pair (x, p) satisfying ϵ-CS, we consider the arc set $\mathcal{A}^*$ that contains all push list arcs oriented in the direction of flow change. In particular, for each arc (i, j) in the forward portion $L^+(i)$ of the push list of a node i, we introduce an arc (i, j) in $\mathcal{A}^*$ and for each arc (j, i) in the backward portion $L^-(i)$ of the push list of node i we introduce an arc (i, j) in $\mathcal{A}^*$ (thus the direction of the latter arc is reversed). The set of nodes $\mathcal{N}$ and the set $\mathcal{A}^*$ define the *admissible graph* $G^* = (\mathcal{N}, \mathcal{A}^*)$ [BPT95]. Note that an arc can be in the push list of at most one node, so G^* is well defined. We will consider methods that keep G^* acyclic at all iterations. Intuitively, because we move flow in the direction of the arcs in G^*, keeping G^* acyclic helps to limit the number of flow adjustments between price rises. To ensure that initially the admissible graph is acyclic, one possibility is to choose, for any initial price vector p^0, the initial flow vector x^0 such that (x^0, p^0) satisfies 0-CS, that is,

$$f_{ij}^-(x_{ij}^0) \le p_i^0 - p_j^0 \le f_{ij}^+(x_{ij}^0), \qquad \forall\ (i, j) \in \mathcal{A}. \tag{14}$$

It can be seen that this choice is always possible [BPT95], that ϵ-CS is satisfied by (x^0, p^0) for any $\epsilon > 0$, and that the initial admissible graph is empty and thus acyclic.

In the next two sections, we will study two specializations of the generic auction method. These methods perform flow adjustment by moving flow out of nodes with positive surplus along push-list arcs and they perform price rises only on nodes with empty push lists. In addition, they keep the admissible graph acyclic at all iterations and have favorable complexity bounds. As a final note, we saw earlier that, upon termination of the generic auction method, the flow vector and price vector satisfy ϵ-CS and the flow vector is feasible, so they are approximately optimal for (P) and (D). The following proposition, due to [BHT87], makes this notion of approximate optimality more precise.

Proposition 4: For each $\epsilon > 0$, let $x(\epsilon)$ and $p(\epsilon)$ denote any flow and price vector pair satisfying ϵ-CS with $x(\epsilon)$ feasible and let $\xi(\epsilon)$ denote any flow vector satisfying CS together with $p(\epsilon)$ (note that $\xi(\epsilon)$ need not be feasible). Then

$$0 \leq f\big(x(\epsilon)\big) + q\big(p(\epsilon)\big) \leq \epsilon \sum_{(i,j)\in\mathcal{A}} |x_{ij}(\epsilon) - \xi_{ij}(\epsilon)| \,.$$

Furthermore, $f\big(x(\epsilon)\big) + q\big(p(\epsilon)\big) \to 0$ as $\epsilon \to 0$.

Proposition 4 does not give an a priori estimate of how small ϵ has to be in order to achieve a certain degree of approximate optimality as measured by the duality gap. However, in the common case where finiteness of the arc cost functions f_{ij} imply lower and upper bounds on the arc flows, Prop. 4 does yield such an estimate for ϵ.

3. THE ϵ-RELAXATION METHOD

In this section we consider the ϵ-relaxation method presented in [BPT95] for solving (P) and (D). We will see that it is a special case of the generic auction method and, as such, its complexity may be analyzed using the results of Section 2. First, we describe the method.

For a fixed $\epsilon > 0$ and $\beta \in (0,1)$, and a given flow-price vector pair (x,p) satisfying ϵ-CS, an iteration of the ϵ-relaxation method updates (x,p) as follows:

An Iteration of the ϵ-Relaxation Method

Step 1: Select a node i with positive surplus g_i (see Eq. (2)); if no such node exists, terminate the method.

Step 2: If the push list of i is empty, go to Step 3. Otherwise, choose an arc from the push list of i and perform a *δ-flow push* towards the opposite node j (i.e., increase f_{ij} by δ if (i,j) is the arc; decrease f_{ji} by δ if (j,i) is the arc), where

$$\delta = \min\{g_i, \text{ flow margin of the chosen arc}\}.$$

If the surplus of i becomes zero, go to the next iteration; otherwise, go to Step 2.

Step 3: Increase the price p_i by the maximum amount that maintains ϵ-CS. Go to the next iteration.

To see that the ϵ-relaxation method is a specialization of the generic auction method of Section 2, simply notice that Step 3 is a price rise on node i and that Step 2 adjusts the flows in such a way that ϵ-CS is maintained and nodes with nonnegative surplus continue to have nonnegative surplus for all subsequent iterations. The reason for the latter is that a flow push at a node i cannot make the surplus of i negative (by choice of δ in Step 2), and cannot decrease the surplus of neighboring nodes. Furthermore, the ϵ-relaxation method performs a price rise only on nodes with empty push list. Then, by Prop. 3, each price rise increment is at least $\beta\epsilon$ and, by Prop. 1, the number of price rises (i.e., Step 3) on each node is at most $(K+1)(N-1)/\beta$, where K is any nonnegative scalar such that the initial price vector satisfies $K\epsilon$-CS together with some feasible flow vector. Thus, to prove finite termination of the ϵ-relaxation method, it suffices to show that the number of flow pushes (i.e., Step 2) performed between successive price rises is finite. Following [BPT95], we show this by first showing that the method keeps the admissible graph acyclic.

Proposition 5: If initially the admissible graph is acyclic, then the admissible graph remains acyclic at all iterations of the ϵ-relaxation method.

Proof: We use induction. Initially, the admissible graph G^* is acyclic by assumption. Assume that G^* remains acyclic for all subsequent iterations up to the mth iteration for some m. We will prove that after the mth iteration G^* remains acyclic. Clearly, after a flow push the admissible graph remains acyclic, since it either remains unchanged, or some arcs are deleted from it. Thus we only have to prove that after a price rise on a node i, no cycle involving i is created. We note that, after a price rise on node i, all incident arcs to i in the admissible graph at the start of the mth iteration are deleted and new arcs incident to i are added. We claim that i cannot have any incoming arcs which belong to the admissible graph. To see this, note that, just before a price rise on node i, we have from (4) that

$$p_j - p_i - f_{ji}^+(x_{ji}) \le \epsilon, \qquad \forall\ (j,i) \in \mathcal{A},$$

and since each price rise is at least $\beta\epsilon$, we must have

$$p_j - p_i - f_{ji}^+(x_{ji}) \le (1-\beta)\epsilon, \qquad \forall\ (j,i) \in \mathcal{A},$$

after the price rise. Then, by Eq. (13a), (j,i) cannot be in the push list of node j. By a similar argument, we have that (i,j) cannot be in the push list of j for all $(i,j) \in \mathcal{A}$. Thus, after a price rise on i, node i cannot have any incoming arcs belonging to the admissible graph, so no cycle involving i can be created. **Q.E.D.**

We say that a node i is a *predecessor* of a node j in the admissible graph G^* if a directed path (i.e., a path having no backward arc) from i to j exists in G^*. Node j is then called a *successor* of i. Observe that, in the ϵ-relaxation method, flow is pushed towards the successors of a node and if G^* is acyclic, flow cannot be pushed from a node to any of its predecessors. A δ-flow push along an arc in $\mathcal{A}$ is said to be *saturating* if δ is equal to the flow margin of the arc. By our choice of δ in the ϵ-relaxation method, a nonsaturating flow push always exhausts (i.e., sets to zero) the surplus of the starting node of the arc. Then, by using Prop. 5, we obtain the following result as in [BPT95, Prop. 5].

Proposition 6: If initially the admissible graph is acyclic, then the number of flow pushes between two successive price rises (not necessarily at the same node) performed by the ϵ-relaxation method is finite.

Proof: We observe that a saturating flow push along an arc removes the arc from the admissible graph, while a nonsaturating flow push does not add a new arc to the admissible graph. Thus the number of saturating flow pushes that can be performed between successive price rises is at most A. It will thus suffice to show that the number of nonsaturating flow pushes that can be performed between saturating flow pushes is finite. Assume the contrary, that is, there is an infinite sequence of successive nonsaturating flow pushes, with no intervening saturating flow push. Then the admissible graph remains fixed throughout this sequence. Furthermore, the surplus of some node i^0 must be exhausted infinitely often during this sequence. This can happen only if the surplus of some predecessor i^1 of i^0 is exhausted infinitely often during the sequence. Continuing in this manner we construct an infinite succession of predecessor nodes $\{i^k\}_{k=0,1,\ldots}$. Thus some node in this sequence must be repeated, which is a contradiction since the admissible graph is acyclic. **Q.E.D.**

By refining the proof of Prop. 6, we can further show that the number of flow pushes between successive price rises is at most $(N+1)A$, from which a complexity bound for the ϵ-relaxation method may be readily derived. Below, we consider an implementation of the method, also presented in [BPT95, Section 4], that has a very good complexity.

Efficient Implementations

Here we consider a particularly efficient implementation, called the *sweep implementation*, of the ϵ-relaxation method. This implementation was introduced in [Ber86a] and was analyzed in more detail in [BeE88], [BeT89], and [BeC93] for the linear cost case. We will analyze the running time of this implementation for the general convex cost case. The analysis was orginally presented in the Ph.D. thesis of the second author [Pol95] and in the subsequent paper [BPT95]. Here we only review the basic ideas and the main results, some of which will also be used to analyze the auction/sequential-shortest-path algorithm of the next section. The reader is referred to the above thesis and paper for more details of the analysis and the proofs.

In the sweep implementation of the ϵ-relaxation method, the admissible graph is acyclic initially (and, by Prop. 5, it remains acyclic at all iterations), and the nodes are chosen in Step 1 of the iteration in an order which we now describe: All the nodes are kept in a linked list T, which is traversed from the first to the last element. The order of the nodes in the list is consistent with the successor order implied by the admissible graph; that is, if a node j is a successor of a node i, then j must appear after i in the list. If the initial admissible graph is empty, as is the case with the initialization of Eq. (14), the initial list is arbitrary. Otherwise, the initial list must be consistent with the successor order of the initial admissible graph. The list is updated in a way that maintains the consistency with the successor order. In particular, let i be the node chosen in Step 1 of the iteration, and let N_i be the subset of nodes of T that are after i in T. If the price of i changes in this iteration, then node i is removed from its position in T and placed in the first position of T. The node chosen in the next iteration, if N_i is nonempty, is the node $i' \in N_i$ with positive surplus which ranks highest in T. Otherwise, the positive surplus node ranking highest in T is chosen. It can be shown (see the references cited earlier) that, with this rule of repositioning the nodes following a price change, the list order is consistent with the successor order implied by the admissible graph at all iterations. The idea of the sweep implementation is that an ϵ-relaxation iteration at a node i that has predecessors with positive surplus may be wasteful, since the surplus of i will be set to zero and become positive again through a flow push at a predecessor node.

The next proposition gives a bound on the number of flow pushes made by the sweep implementation of the ϵ-relaxation method. This result is based on the observations that (i) between successive saturating flow pushes on an arc, there is at least one price rise performed on one of the end nodes of the arc; (ii) between successive price rises (not necessarily at the same node), the number of nonsaturating flow pushes is at most N. We refer the reader to [BPT95, Props. 7 and 8] for a detailed proof of this result.

Proposition 7: Let K be any nonnegative scalar such that the initial price vector for the sweep implementation of the ϵ-relaxation method satisfies $K\epsilon$-CS together with some feasible flow vector. Then, the number of price rises on each node, the number of saturating flow pushes, and the number of nonsaturating flow pushes up to termination of the method are $O(KN)$, $O(KNA)$, and $O(KN^3)$, respectively.

By using Prop. 7, we now bound the running time for the sweep implementation of the ϵ-relaxation method. The dominant computational requirements are:

(1) The computation required for price rises.

(2) The computation required for saturating flow pushes.

(3) The computation required for nonsaturating flow pushes.

In contrast to the linear cost case, we cannot express the running time in terms of the size of the problem data since the latter is not well defined for convex cost functions. Instead, we introduce a set of simple operations performed by the ϵ-relaxation method, and we estimate the number of these operations. In particular, in addition to the usual arithmetic operations with real numbers, we consider the following operations:

(a) Given the flow x_{ij} of an arc (i,j), calculate the cost $f_{ij}(x_{ij})$, the left derivative $f_{ij}^-(x_{ij})$, and the right derivative $f_{ij}^+(x_{ij})$.

(b) Given the price differential $t_{ij} = p_i - p_j$ of an arc (i,j), calculate $\sup\{\xi \mid f_{ij}^+(\xi) \leq t_{ij}\}$ and $\inf\{\xi \mid f_{ij}^-(\xi) \geq t_{ij}\}$.

Operation (a) is needed to compute the push list of a node and a price increase increment; operation (b) is needed to compute the flow margin of an arc and the flow initialization of Eq. (14). Complexity will thus be measured in terms of the total number of operations performed by the method, as is stated in the following proposition as a consequence of Prop. 7.

Proposition 8: Let K be any nonnegative scalar such that the initial price vector for the sweep implementation of the ϵ-relaxation method satisfies $K\epsilon$-CS together with some feasible flow vector. Then, the method requires $O(KN^3)$ operations up to termination.

The theoretical and the practical performance of the ϵ-relaxation method can be further improved by a technique known as *ϵ-scaling*, originally conceived in [Ber79] as a means of improving the performance of the auction algorithm for the assignment problem and later used in [Gol87] and [GoT90] for improving the complexity of related algorithms for linear cost network flow. The idea of ϵ-scaling is to apply the ϵ-relaxation method several times, starting with a large value of ϵ, say ϵ^0, and to successively reduce ϵ (typically at a geometric rate) up to a final value, say $\overline{\epsilon}$, that will give the desirable degree of accuracy to our solution. Furthermore, the price and flow information from one application of the method is passed to the next. The ϵ-scaling implementation of the ϵ-relaxation method is described and analyzed in detail in [BPT95, Section 4]. In particular, it is shown there that if ϵ^0 is chosen sufficiently large so that the initial price vector satisfies ϵ^0-CS together with some feasible flow vector, then the running time of the ϵ-relaxation method using the sweep implementation and ϵ-scaling is $O\big(N^3 \ln(\epsilon^0/\overline{\epsilon})\big)$ operations.

We note that a complexity bound of $O\big(NA\ln(N)\ln(\epsilon^0/\bar{\epsilon})\big)$ operations was derived in [KaM93] for the tighten and cancel method. For relatively dense network flow problems where $A = \Theta(N^2/\ln N)$, our complexity bound for the ϵ-relaxation method is more favorable, while for sparse problems, where $A = \Theta(N)$, the reverse is true. Also, it may be possible to obtain sharper complexity bounds for special (but still interesting) classes of problems, such as those involving quadratic arc cost functions, and this is a subject for further research.

4. THE AUCTION/SEQUENTIAL-SHORTEST-PATH (ASSP) ALGORITHM

The auction/sequential-shortest-path (ASSP) algorithm was proposed in [Ber92] for linear cost network flow problems. In this section, we consider an extension of this algorithm to the general convex cost case of (P) and (D). The resulting ASSP algorithm is a special case of the generic auction method and, as such, we will analyze its complexity by using the results of Section 2 and by adapting the analysis of Section 3. This algorithm differs from the ϵ-relaxation method of Section 3 in that, instead of pushing flow along a push-list arc to any node, it pushes flow along a path of push-list arcs to a node with negative surplus. In fact, whereas a flow push in the ϵ-relaxation method may increase the surplus of a node in magnitude (e.g., when flow is pushed to a neighboring node with nonnegative surplus), the ASSP algorithm maintains the surplus of each node to be nonincreasing in magnitude.

First, we introduce some definitions that are needed to describe the ASSP algorithm. For a path P in $(\mathcal{N}, \mathcal{A})$, we denote by $s(P)$ and $t(P)$ the starting node and the terminal node, respectively, of P. We define two operations on a given path $P = (n_1, n_2, \ldots, n_k)$: A *contraction* of P deletes the terminal node of P and the arc incident to this node. An *extension* of P by an arc (n_k, n_{k+1}) or an arc (n_{k+1}, n_k), replaces P by the path $(n_1, n_2, \dots, n_k, n_{k+1})$ and adds to P the corresponding arc. For any $\epsilon > 0$ and $\beta \in (0,1)$, and any flow-price vector pair (x,p) satisfying ϵ-CS, we say that a path P in $(\mathcal{N}, \mathcal{A})$ is *augmenting* if each forward (respectively, backward) arc (i,j) of P is in the push list of i (respectively, j) and $s(P)$ is a *source* (i.e., has positive surplus) and $t(P)$ is a *sink* (i.e., has negative surplus). Below we describe the ASSP algorithm for solving (P) and (D).

For a fixed $\epsilon > 0$ and $\beta \in (0,1)$, and a given flow-price vector pair (x,p) satisfying ϵ-CS, an iteration of the ASSP algorithm updates (x,p) as follows:

An Iteration of the ASSP Algorithm

Step 1: Select a node i with positive surplus g_i (see Eq. (2)) and let the path P consist of only this node; if no such node exists, terminate the algorithm.

Step 2: Let i be the terminal node of the path P. If the push list of i is empty, then go to Step 3; otherwise, go to Step 4.

Step 3 **(Contract Path):** Increase the price p_i by the maximum amount that maintains ϵ-CS. If $i \neq s(P)$, contract P. Go to Step 2.

Step 4 **(Extend Path):** Select an arc (i,j) (or (j,i)) from the push list of i and extend P by this arc. If the surplus of j is negative, go to Step 5; otherwise, go to Step 2.

Step 5 **(Augmentation):** Perform an *augmentation* along the path P by the amount δ (i.e., increase the flow of all forward arcs in P and decrease the flow of all backward arcs in P by δ), where

$$\delta = \min \left\{ g_{s(P)}, -g_{t(P)}, \text{minimum of flow margins of the arcs of } P \right\}.$$

Go to the next iteration.

Roughly speaking, at each iteration of the ASSP algorithm, the path P starts as a single source and is successively extended or contracted until the terminal node of P is a sink. Then an augmentation along P is performed so to decrease (respectively, increase) the surplus of the starting node (respectively, terminal node) towards zero, while leaving the surplus of the remaining nodes unchanged. In case of a contraction, the price of the terminal node of P is strictly increased. To see that the ASSP algorithm is a specialization of the generic auction method of Section 2, notice that Step 2 is a price rise on node i and that Step 5 adjusts the flows in such a way that ϵ-CS is maintained and nodes with nonnegative surplus continue to have nonnegative surplus for all subsequent iterations. The reason for the latter is that an augmentation along P changes the surplus of only two nodes $s(P)$ and $t(P)$, and, by our choice of δ, the surplus of the node $s(P)$ remains nonnegative after the augmentation. Furthermore, the ASSP algorithm performs price rise only on nodes with empty push list. Then, by Prop. 3, each price rise increment is at least $\beta\epsilon$ and, by Prop. 1, the number of price rises (i.e., path contractions) on each node is at most $(K+1)(N-1)/\beta$, where K is any nonnegative scalar such that the initial price vector satisfies $K\epsilon$-CS together with some feasible flow vector. Thus, to prove finite termination of the ASSP algorithm, it suffices to show that the number of path extensions (i.e., Step 4) and the number of augmentations (i.e., Step 5) performed between successive path contractions is finite. Similar to Section 3, we show this by first showing that the algorithm keeps the admissible graph acyclic and that the path P, when its backward arcs are reversed in direction, belongs to the admissible graph.

Proposition 9: If initially the admissible graph is acyclic, then the admissible graph remains acyclic at all iterations of the ASSP algorithm. Moreover, the path P maintained by the algorithm, when its backward arcs are reversed in direction, belongs to the admissible graph at all times.

Proof: The admissible graph can change either by a price rise (Step 3) or by an augmentation (Step 5). An augmentation keeps the admissible graph acyclic because, after an augmentation, the admissible graph either remains unchanged or some arcs are deleted from it. A price rise keeps the admissible graph acyclic, as was shown in the proof of Prop. 5.

To show that P, when its backward arcs are reversed in direction, belongs to the admissible graph at all times, we simply observe that a path extension maintains this property (since the arc added to P is in the push list of the terminal node of P) and that a path contraction also maintains this property (since a price rise on the terminal node of P changes the admissible graph only by adding/deleting arcs incident to this node and, after the contraction, this node and its incident arc in P are both deleted from P). **Q.E.D.**

By using Prop. 9, we have the following result that gives a bound on the number of augmentations and path extensions performed by the ASSP algorithm between successive path contractions. By using this bound and the bound on the number of path contractions found earlier, we can readily derive a complexity bound for the ASSP algorithm.

Proposition 10: If initially the admissible graph is acyclic, then the number of augmentations and path extensions between two successive path contractions (not necessarily at the same node) performed by the ASSP algorithm are at most $A + N$ and $N(A + N)$, respectively.

Proof: We observe that an augmentation does not increase the number of nodes with nonzero surplus and does not add any arc to the admissible graph. Moreover, after an augmentation, either an arc is removed from the admissible graph or a node has its surplus set to zero. Thus, the number of arcs in the admissible graph plus the number of nodes with nonzero surplus is decreased by at least one after each augmentation. It follows that the number of augmentations between successive path contractions is at most $A + N$.

By Prop. 9, the path P always belongs to the admissible graph which is acyclic, so P cannot have repeated nodes and hence the number of successive extensions of P (before a contraction or an augmentation is performed) is at most N. Thus, the number of path extensions between successive path contractions is at most $N \cdot$ (number of augmentations between successive path contractions) $\leq N(A + N)$. **Q.E.D.**

There is an interesting connection between the ASSP algorithm and the auction algorithm of [Ber91] for finding a shortest path between two nodes, which explains our use of the name ASSP. In particular, we note that each iteration comprises a sequence of path extensions and contractions, followed by an augmentation at the end. Let us fix an iteration and let $(\hat{x}, \hat{p})$ be the flow-price vector at the start of this iteration. Let us now define an arc set $\mathcal{A}_{\mathcal{R}}$ by introducing, for each arc $(i,j) \in \mathcal{A}$, two arcs in $\mathcal{A}_{\mathcal{R}}$: an arc (i,j) with length $\hat{p}_j - \hat{p}_i + f_{ij}^+(\hat{x}_{ij}) + \epsilon$ and an arc (j,i) with length $\hat{p}_i - \hat{p}_j - f_{ij}^-(\hat{x}_{ij}) + \epsilon$. The resulting graph $G_{\mathcal{R}} = (\mathcal{N}, \mathcal{A}_{\mathcal{R}})$ will be referred to as the *reduced graph.* Note that, because the pair $(\hat{x}, \hat{p})$ satisfies ϵ-CS, the arc lengths in the reduced graph are nonnegative. Furthermore, the reduced graph contains no zero-length cycles whenever the admissible graph is acyclic (since such a cycle would belong to the admissible graph). It can then be verified that the sequence of path extensions and contractions performed during the iteration is just the algorithm of [Ber91] applied to find a shortest path in the reduced graph $G_{\mathcal{R}}$ from a given source to any sink.

5 COMPUTATIONAL EXPERIMENTATION

We have developed and tested two experimental Fortran codes implementing the methods of Sections 3 and 4, with $\beta = 1/2$, for convex quadratic cost problems. The first code, named NE-RELAX-F, implements the ϵ-relaxation method with the sweep implementation and ϵ-scaling as described in Section 3 (also see [BPT95] for alternative implementations). The second code, named ASSP-N, implements the auction/sequential-shortest-path algorithm with some enhancements described in [Pol95]. These codes are based on corresponding codes for linear cost problems described in Appendix 7 of [Ber91], which have been shown to be quite efficient. Several changes and enhancements were introduced in our codes to handle quadratic costs. In particular, all computations are done in real rather than integer arithmetic, and ϵ-scaling, rather than arc cost scaling, is used.

The codes NE-RELAX-F and ASSP-N were compared to two existing Fortran codes NRELAX and MNRELAX from [BHT87]. The latter implement the relaxation method for, respectively, strictly convex quadratic cost and convex quadratic cost problems, and are believed to be quite efficient. All codes were compiled and run on a Sun Sparc-5 workstation with 24 megabytes of RAM under the Solaris operating system. We used the -O compiler option in order to take advantage of the floating point unit and the design characteristics of the Sparc-5 processor. Unless otherwise indicated, all codes upon termination meet the criterion that the node surpluses are below 10^{-5} in magnitude and the cost of the flow vector and the cost of the price vector agree in their first 12 digits.

For our test problems, the cost functions are of the form

$$f_{ij}(x_{ij}) = \begin{cases} a_{ij}x_{ij} + b_{ij}x_{ij}^2 & \text{if } 0 \le x_{ij} \le c_{ij}, \\ \infty & \text{otherwise,} \end{cases}$$

for some $a_{ij} \in \Re$ and $b_{ij} \in [0, \infty)$ and $c_{ij} \in [0, \infty)$. We call a_{ij}, b_{ij}, and c_{ij} the linear cost coefficient, the quadratic cost coefficient, and the capacity, respectively, of arc (i, j). We created the test problems using two Fortran problem generators. The first is the public-domain generator NETGEN, written by Klingman, Napier and Stutz [KNS74], which generates linear-cost assignment/transportation/transshipment problems having a certain random structure. The second is the generator CHAINGEN, written by the second author, which generates transshipment problems having a chain structure as follows: starting with a chain through all the nodes (i.e., a directed graph with nodes $1, ..., N$ and arcs $(1, 2), (2, 3), ..., (N - 1, N), (N, 1)$), a user-specified number of forward arcs are added to each node (for example, if the user specifies 3 additional arcs per node then the arcs $(i, i + 2)$, $(i, i + 3)$, $(i, i + 4)$ are added for each node i) and, for a user-specified percentage of nodes i, a reverse arc $(i, i - 1)$ is also added. The graphs thus created have long diameters and earlier tests on linear cost problems showed that the created problems are particularly difficult for all methods tested. As the above two generators create only linear-cost problems, we modified the created problems as in [BHT87] so that, for a user-specified percent of the arcs, a nonzero quadratic cost coefficient is generated in a user-specified range.

Our tests were designed to study two key issues:

(a) The performance of the ϵ-relaxation method and the ASSP algorithm relative to the earlier relaxation methods, and the dependence of this performance on network topology and problem ill-conditioning.

(b) The sensitivity of the ϵ-relaxation method and the ASSP algorithm to problem ill-conditioning.

Ill-conditioned problems were created by assigning to some of the arcs much smaller (but nonzero) quadratic cost coefficients compared to other arcs. When the arc cost functions have this structure, ill-conditioning in the traditional sense of unconstrained nonlinear programming tends to occur.

We experimented with three sets of test problems: the first set comprises well-conditioned strictly convex quadratic cost problems generated using NETGEN (Table 1); the second set comprises ill-conditioned strictly convex quadratic cost problems and mixed linear/quadratic cost problems generated using NETGEN (Table 3); the third set comprises well-conditioned strictly convex quadratic cost problems generated using CHAINGEN (Table 5). The running time of the codes on these problems are shown in the last three to four columns of Tables 2, 4, and 6. On the ill-conditioned NETGEN problems and the CHAINGEN problems, NRELAX often had difficulty meeting the termination criterion and was terminated early. From the running times we can see that the codes NE-RELAX-F and ASSP-N consistently outperform, by a factor of at least 3 and often much more, the relaxation codes NRELAX and MNRELAX on all test problems, independent of network topology and problem ill-conditioning. In fact, on the CHAINGEN problems, the ϵ-relaxation and auction codes outperform the relaxation codes by an order of magnitude or more.

REFERENCES

[Ber79] Bertsekas, D. P., "A Distributed Algorithm for the Assignment Problems," Laboratory for Information and Decision Systems Working Paper, M.I.T., Cambridge, MA, 1979.

[Ber86a] Bertsekas, D. P., "Distributed Relaxation Methods for Linear Network Flow Problems," Proceedings of 25th IEEE Conference on Decision and Control, 1986, pp. 2101-2106.

[Ber86b] Bertsekas, D. P., "Distributed Asynchronous Relaxation Methods for Linear Network Flow Problems," Laboratory for Information and Decision Systems Report P-1606, M.I.T., Cambridge, MA, 1986.

[Ber91] Bertsekas, D. P., Linear Network Optimization: Algorithms and Codes, M.I.T. Press, Cambridge, MA, 1991.

[Ber92] Bertsekas, D. P., "An Auction/Sequential Shortest Path Algorithm for the Min Cost Flow Problem," Laboratory for Information and Decision Systems Report P-2146, M.I.T., Cambridge, MA, 1992.

[BeC93] Bertsekas, D. P., Castañon, D. A., "A Generic Auction Algorithm for the Minimum Cost Network Flow Problem," Computational Optimization and Applications, Vol. 2, pp. 229-260, 1993.

[BCEZ95] Bertsekas, D. P., Castañon, D. A., Eckstein, J., and Zenios, S. A., "Parallel Computing in Network Optimization", Handbooks in Operations Research and Management Science, (M. O. Ball, et. al, Eds.), Vol. 7, 1995, pp. 331-399.

[BeE87] Bertsekas, D. P., and Eckstein, J., "Distributed Asynchronous Relaxation Methods for Linear Network Flow Problems," Proceedings of IFAC '87, Munich, Germany, July 1987.

[BeE88] Bertsekas, D. P., and Eckstein, J., "Dual Coordinate Step Methods for Linear Network Flow Problems," Mathematical Programming, Vol. 42, 1988, pp. 203-243.

[BeE87] Bertsekas, D. P., and El Baz, D., "Distributed Asynchronous Relaxation Methods for Convex Network Flow Problems," SIAM Journal on Control and Optimization, Vol. 25, 1987, pp. 74-85.

[BHT87] Bertsekas, D. P., Hosein, P. A., and Tseng, P., "Relaxation Methods for Network Flow Problems with Convex Arc Costs," SIAM Journal on Control and Optimization, Vol. 25, 1987, pp. 1219-1243.

[BPT95] Bertsekas, D. P., Polymenakos, L. C., and Tseng, P., "An ϵ-Relaxation Method for Separable Convex Cost Network Flow Problems," Laboratory for Information and Decision Systems Report LIDS-P-2299, M.I.T., Cambridge, MA, 1995; to appear in SIAM Journal on Optimization.

[BeT89] Bertsekas, D. P., and Tsitsiklis, J. N., Parallel and Distributed Computation: Numerical Methods, Prentice-Hall, Englewood Cliffs, NJ, 1989.

[DMZ95] De Leone, R., Meyer, R. R., and Zakarian, A., "An ϵ-Relaxation Algorithm for Convex Network

Flow Problems," Computer Sciences Department Technical Report, University of Wisconsin, Madison, WI, 1995.

[FoF62] Ford, L. R., Jr., and Fulkerson, D. R., Flows in Networks, Princeton University Press, Princeton, NJ, 1962

[Gol87] Goldberg, A. V., "Efficient Graph Algorithms for Sequential and Parallel Computers," Laboratory for Computer Science Technical Report TR-374, M.I.T., Cambridge, MA, 1987.

[GoT90] Goldberg, A. V., and Tarjan, R. E., "Solving Minimum Cost Flow Problems by Successive Approximation," Mathematics of Operations Research, Vol. 15, 1990, pp. 430-466.

[Hag92] Hager, W. W., "The Dual Active Set Algorithm," in Advances in Optimization and Parallel Computing, Edited by P. M. Pardalos, North-Holland, Amsterdam, Netherland, 1992, pp. 137-142.

[HaH93] Hager, W. W., and Hearn, D. W., "Application of the Dual Active Set Algorithm to Quadratic Network Optimization," Computational Optimization and Applications, Vol. 1, 1993, pp. 349-373.

[KaM84] Kamesam, P. V., and Meyer, R. R., "Multipoint Methods for Separable Nonlinear Networks," Mathematical Programming Study, Vol. 22, 1984, pp. 185-205.

[KaM93] Karzanov, A. V., and McCormick, S. T., "Polynomial Methods for Separable Convex Optimization in Unimodular Linear Spaces with Applications to Circulations and Co-circulations in Network," Faculty of Commerce Report, University of British Columbia, Vancouver, BC, 1993; to appear in SIAM Journal on Computing.

[KNS74] Klingman, D., Napier, A., and Stutz, J., "NETGEN - A Program for Generating Large Scale (Un) Capacitated Assignment, Transportation, and Minimum Cost Flow Network Problems," Management Science, Vol. 20, 1974, pp. 814-822.

[LiZ94] Li, X., and Zenios, S. A., "Data Parallel Solutions of Min-Cost Network Flow Problems Using ϵ-Relaxations," European Journal of Operational Research, Vol. 79, 1994, pp. 474-488.

[Mey79] Meyer, R. R., "Two-Segment Separable Programming," Management Science, Vol. 25, 1979, pp. 285-295.

[NiZ93] Nielsen, S. S., and Zenios, S. A., "On the Massively Parallel Solution of Linear Network Flow Problems," in Network Flow and Matching: First DIMACS Implementation Challenge, Edited by D. Johnson and C. McGeoch, American Mathematical Society, Providence, RI, 1993, pp. 349-369.

[Pol95] Polymenakos, L. C. "ϵ-Relaxation and Auction Algorithms for the Convex Cost Network Flow Problem," Electrical Engineering and Computer Science Department Ph.D. Thesis, M.I.T., Cambridge, MA, 1995.

[Roc70] Rockafellar, R. T., Convex Analysis, Princeton University Press, Princeton, NJ, 1970.

[Roc84] Rockafellar, R. T., Network Flows and Monotropic Programming, Wiley-Interscience, New York, NY, 1984.

[TBT90] Tseng, P., Bertsekas, D. P., and Tsitsiklis, J. N., "Partially Asynchronous, Parallel Algorithms for Network Flow and Other Problems," SIAM Journal on Control and Optimization, Vol. 28, 1990, pp. 678-710.

[Ven91] Ventura, J. A., "Computational Development of a Lagrangian Dual Approach for Quadratic Networks," Networks, Vol. 21, 1991, pp. 469-485.

[Wei74] Weintraub, A., "A Primal Algorithm to Solve Network Flow Problems with Convex Costs," Management Science, Vol. 21, 1974, pp. 87-97.

Problem Name	Nodes	Arcs	Linear Cost	Quad Cost	Total Supply	Capacity Range
prob1	200	1300	[1-100]	[5,10]	10000	[100-500]
prob2	200	1500	[1-100]	[5,10]	10000	[100-500]
prob3	200	2000	[1-100]	[5,10]	10000	[100-500]
prob4	200	2200	[1-100]	[5,10]	10000	[100-500]
prob5	200	2900	[1-100]	[5,10]	10000	[100-500]
prob6	300	3150	[1-100]	[5,10]	10000	[100-500]
prob7	300	4500	[1-100]	[5,10]	10000	[100-500]
prob8	300	5155	[1-100]	[5,10]	10000	[100-500]
prob9	300	6075	[1-100]	[5,10]	10000	[100-500]
prob10	300	6300	[1-100]	[5,10]	10000	[100-500]
prob11	400	1500	[1-100]	[5,10]	10000	[100-500]
prob12	400	2250	[1-100]	[5,10]	10000	[100-500]
prob13	400	3000	[1-100]	[5,10]	10000	[100-500]
prob14	400	3750	[1-100]	[5,10]	10000	[100-500]
prob15	400	4500	[1-100]	[5,10]	10000	[100-500]
prob16	400	1306	[1-100]	[5,10]	10000	[100-500]
prob17	400	2443	[1-100]	[5,10]	10000	[100-500]
prob18	400	1416	[1-100]	[5,10]	10000	[100-500]
prob19	400	2836	[1-100]	[5,10]	10000	[100-500]
prob20	400	1382	[1-100]	[5,10]	10000	[100-500]
prob21	400	2676	[1-100]	[5,10]	10000	[100-500]
prob22	1000	3000	[1-100]	[5,10]	10000	[1000-2000]
prob23	1000	5000	[1-100]	[5,10]	10000	[1000-2000]
prob24	1000	10000	[1-100]	[5,10]	10000	[1000-2000]

Table 1: The NETGEN problems with all arcs having quadratic cost coefficients in the range shown. The problems prob1-prob17 are identical to the problems 1-17 of Table 1 of [BHT87]. The problems named prob18, prob19, prob20, prob21 correspond to the problems 20, 23, 24, 25, respectively, of Table 1 of [BHT87].

Problem	NRELAX	MNRELAX	NE-RELAX-F	ASSP-N
prob1	7.95	6.0	1.95	1.09
prob2	7.55	6.35	2.13	1.27
prob3	2.88	5.65	2.13	0.707
prob4	20.45	10.62	2.4	1.42
prob5	2.32	24.8	1.45	1.13
prob6	7.31	22.11	2.71	1.43
prob7	7.52	21.12	3.94	1.69
prob8	48.3	26.72	3.88	2.7
prob9	7.25	22.71	3.22	2.54
prob10	4.41	31.53	2.7	3.02
prob11	69.25	15.07	8.79	4.88
prob12	17.68	17.24	4.98	2.91
prob13	22.00	20.43	7.3	4.14
prob14	13.2	24.3	3.03	2.33
prob15	10.01	35.99	7.42	4.11
prob16	85.10	25.46	8.64	4.87
prob17	31.63	21.52	7.38	4.14
prob18	7.51	9.03	0.96	0.91
prob19	45.43	26.76	8.63	5.07
prob20	79.96	17.71	9.95	7.5
prob21	33.48	23.97	6.8	4.11
prob22	64.42	50.94	8.46	2.44
prob23	26.7	49.06	4.08	4.3
prob24	26	323.296	5.53	5.23

Table 2: Computational Results on a Sun Sparc 5 with 24MB memory. Running times are in seconds.

Problem Name	Nodes	Arcs	Linear Cost	Small Quad Cost	Total Supply	Capacity Range
prob1	200	1300	[1-100]	1	1000	[100-300]
prob2	200	1300	[1-100]	0.1	1000	[100-300]
prob3	200	1300	[1-100]	0.01	1000	[100-300]
prob4	200	1300	[1-100]	0.001	1000	[100-300]
prob5	200	1300	[1-100]	0.0001	1000	[100-300]
prob6	200	1300	[1-100]	0	1000	[100-300]
prob7	400	4500	[1-100]	1	1000	[100-300]
prob8	400	4500	[1-100]	0.1	1000	[100-300]
prob9	400	4500	[1-100]	0.01	1000	[100-300]
prob10	400	4500	[1-100]	0.001	1000	[100-300]
prob11	400	4500	[1-100]	0.0001	1000	[100-300]
prob12	400	4500	[1-100]	0	1000	[100-300]

Table 3: The NETGEN problems with half of the arcs having quadratic cost coefficient in the range [5,10] and the remaining arcs having the small quadratic coefficient indicated. The problems prob6 and prob12 are mixed linear/quadratic cost problems where half of the arcs have quadratic cost coefficient in the range [5,10] and the remaining arcs have zero quadratic cost coefficient.

Problem	NRELAX	MNRELAX	NE-RELAX-F	ASSP-N
prob1	3.6	3.6	0.5	0.50
prob2	20.95	4.3	0.61	0.53
prob3	56.1	3.6	0.67	0.62
prob4	(5)791.24	3.28	0.73	0.67
prob5	(5)1866.67	2.7	0.77	0.94
prob6	-	2.23	0.69	0.67
prob7	52.22	14.1	1.73	1.53
prob8	53.42	11.26	1.88	1.42
prob9	(5)80.5	13.76	2.3	1.56
prob10	(5)710.73	15.0	2.67	2.0
prob11	(4)5753.45	13.56	3.67	3.4
prob12	-	8.33	2.79	2.51

Table 4: Computational Results on a Sun Sparc 5 with 24MB memory. On the very ill-conditioned problems, NRELAX was taking extremely long running times and was terminated early. The numbers in parentheses indicate the number of significant digits of accuracy of the answer given by NRELAX upon termination. The running times of NE-RELAX-F and ASSP-N on the mixed linear/quadratic cost problems prob6 and prob12 are included to demonstrate the fact that these methods are not affected significantly by ill-conditioning.

Problem Name	Nodes	Linear Cost	Total Supply	Capacity Range	Add. Forw. Arcs	Total # Arcs
prob1	50	[1-100]	1000	[100-1000]	4	269
prob2	100	[1-100]	1000	[100-1000]	4	544
prob3	150	[1-100]	1000	[100-1000]	4	819
prob4	200	[1-100]	1000	[100-1000]	4	1094
prob5	250	[1-100]	1000	[100-1000]	4	1369
prob6	300	[1-100]	1000	[100-1000]	6	2235
prob7	350	[1-100]	1000	[100-1000]	6	2610
prob8	400	[1-100]	1000	[100-1000]	8	3772
prob9	450	[1-100]	1000	[100-1000]	8	4247
prob10	500	[1-100]	1000	[100-1000]	10	5705

Table 5: The CHAINGEN problems with all arcs having quadratic cost coefficients in the range [5,10]. Half of the nodes have an additional reverse arc.

Problem	MNRELAX	NE-RELAX-F	ASSP-N
prob1	1.19	0.18	0.26
prob2	14.97	0.68	0.9
prob3	15.65	1.22	1.72
prob4	33.03	2.17	2.85
prob5	41.08	2.48	3.52
prob6	93.9	4.6	7.0
prob7	266.9	5.9	7.17
prob8	1102.64	10.4	13.95
prob9	2152.51	10.81	14.6
prob10	>2300	17.72	24.15

Table 6: Computational Results on a Sun Sparc 5 with 24MB memory. On these problems, NRELAX was taking extremely long running times even for 5 digits of accuracy. For this reason, we are not reporting any running times for NRELAX on these problems. MNRELAX also was taking a very long time on the last problem and was terminated early. This is indicated by the > sign.