April 1996

Appeared in Network Optimization, Pardalos, P. M., Hearn, D. W., and Hager, W. W. (eds.), Lecture Notes in Economics and Mathematical Systems, Springer-Verlag, N.Y., 1998, pp. 103-126.

$\mathcal{E}$-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 propose two new methods for the solution of the single commodity, separable convex cost network flow problem: the ϵ-relaxation method and the auction/sequential shortest path method. Both methods were originally developed for linear cost problems and reduce to their linear conterparts when applied to such problems. We show that both methods stem from a common algorithmic framework, that they terminate with a near optimal solution, and we provide an associated complexity analysis. 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., Rm. 35-210, Cambridge, Mass., 02139. Email: dimitrib@mit.edu

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

[4] Department of Mathematics, Univ. of Washington, Seattle, Wash., 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 $\mathcal{A} \subset \mathcal{N} \times \mathcal{N}$. The number of nodes is N and the number of arcs is denoted by A. We denote by x_{ij} the *flow* of the arc (i, j). We refer to the vector $x = \{x_{ij} \mid (i, j) \in \mathcal{A}\}$ as the flow vector. The convex cost network flow problem with separable cost function is defined as

$$\text{minimize} \quad \sum_{(i,j)\in\mathcal{A}} f_{ij}(x_{ij}) \tag{P}$$

$$\text{subject to} \quad \sum_{\{j|(i,j)\in\mathcal{A}\}} x_{ij} - \sum_{\{j|(j,i)\in\mathcal{A}\}} x_{ji} = s_i, \qquad \forall\, i \in \mathcal{N}, \tag{1}$$

where s_i are given scalars, $f_{ij} : \Re \to (-\infty, \infty]$ are given convex, closed, proper functions. We further assume that the functions f_{ij} are extended real-valued, lower semicontinuous, and proper (not identically taking the value ∞). We refer to problem (P) as the *primal* problem. For notational convenience, we have implicitly assumed that there exists at most one arc in each direction between any pair of nodes. 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:

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

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

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

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

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

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} takes the value ∞ outside some compact interval.

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. The standard optimality conditions for a feasible flow-price vector pair (x,p) to be primal and dual optimal are

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

These, known as the *complementary slackness conditions* (CS conditions for short), may be represented explicitly as

$$(x_{ij}, p_i - p_j) \in \Gamma_{ij}, \qquad \forall\ (i,j) \in \mathcal{A},$$

where

$$\Gamma_{ij} = \big\{(x_{ij}, t_{ij}) \in \Re^2 \mid f_{ij}^-(x_{ij}) \le t_{ij} \le f_{ij}^+(x_{ij})\big\}$$

is the *characteristic curve* associated with arc (i,j), as shown in Fig. 1.

There are three classes of methods for solving the problem of this paper for the case of linear arc cost functions: primal, dual and auction methods. The primal and dual methods iteratively improve the primal or the dual cost function. The auction approach 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 may not improve the primal or the dual cost at any iteration, and they are based on a relaxed version of the CS conditions, called ϵ-complementary slackness (ϵ-CS for short). 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 [BCE95], [LiZ91], [NiZ93]). We will extend two such methods, the ϵ-relaxation method and the auction/sequential shortest path method, to the general convex cost case.

One possibility for dealing with the convex cost case is to use efficient ways to reduce the problem to an essentially linear cost problem by piecewise linearization of the arc cost functions; see [Mey79], [KaM84], [Roc84]. Another possibility is to use differentiable unconstrained optimization methods on the dual problem which apply primarily to problems with strictly convex primal cost functions. Such methods are the coordinate descent method [BHT87], the conjugate gradient method [Ven91], and adaptations of other nonlinear programming methods [HaH93], [Hag92], or fixed point methods [BeE87], [TBT90]. A more general alternative, which applies to nondifferentiable dual cost functions as well, is to use an extension of the primal or dual cost improvement methods developed for the linear cost case. In particular, there have been proposals of primal cost improvement methods in [Wei74] and more recently in [KaM93]. There have also been proposals of dual cost improvement methods: the fortified descent method [Roc84] that extends the primal-dual method of Ford and Fulkerson [FoF62], and the relaxation method of [BHT87] that extends the corresponding linear cost relaxation method of [Ber85] and [BeT88]. These methods maintain, together with the price vector, a flow vector that satisfies the ϵ-CS conditions, and progressively work towards primal feasibility. The flow vector becomes feasible at termination.

Figure 1: A cost function f_{ij} and its corresponding characteristic curve.

In this paper we analyze an algorithmic framework for the development of auction algorithms based on the ϵ-CS conditions first introduced in [BHT87] for the case of convex cost arc costs. The ϵ-CS conditions are a relaxed version of the CS coditions and allow more freedom is adjusting the flow-price pair towards feasibility. We then derive and analyze the first extension of an auction method, the ϵ-relaxation method, to the convex arc cost case. It iteratively modifies the price vector, while effecting attendant flow changes that maintain the ϵ-CS conditions. An analogous extension of the auction/sequential shortest path method given in [Ber92] is also derived. It modifies iteratively the price vector finding paths along which it effects flow changes that maintain the ϵ-CS condition. Both methods terminate with a feasible flow-price vector pair which, however, satisfies ϵ-CS rather than CS. They were proposed in the Ph.D. thesis of the second author [Pol95] and they are fundamentally different from the other dual descent methods for nondifferentiable dual cost problems since the price changes are made exclusively along coordinate directions (i.e., one price at a time,) and a price change need not improve the dual cost. Further, upon termination the flow-price pair is optimal only within a factor proportional to ϵ.

The paper is organized as follows. In Section 2 we present the theoretical framework for the development of auction methods. In Section 3, we derive the ϵ-relaxation method extended to solve convex cost problems, we show that this method terminates with a near optimal flow-price vector pair and we review its complexity analysis. In Section 4 we derive the auction/sequential shortest path method extended to the convex cost problems, we prove that the method terminates with a near optimal flow-price vector pair and provide its complexity analysis. The complexity analysis of Section 4 is based on the results of the earlier sections for the ϵ-relaxation method. Finally, in Section 5, we report our computational experience with the methods of Sections 3 and 4 on some convex linear/quadratic cost problems. 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.

2. THE ALGORITHMIC FRAMEWORK

In this section we present a step-by-step construction of a theoretical framework that will lead to auction algorithms, based on the ϵ-CS conditions first introduced in [BHT87] for the case of convex arc costs. A variant of the analysis and the proofs in this section were presented by the authors in [BPT96] in connection with the ϵ-relaxation method for the convex cost network flow problem. The presentation here identifies the general algorithmic framework that leads to the methods we derive in later sections.

The foundation for the new algorithmic framework lies in the ϵ-CS conditions which are defined as a relaxed version the CS conditions. In particular, we say that the flow vector x and the price vector p satisfy ϵ-CS if and only if

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

(see Fig. 2).

Figure 2: A visualization of the ϵ-CS conditions as a cylinder around the characteristic curve (bold line) The shaded area represents flow-price differential pairs that satisfy the ϵ-CS conditions.

As we discussed in Section 1, a feasible flow-price pair is primal and dual optimal if the CS conditions are satisfied. The intuition behind the ϵ-CS conditions is that a feasible flow-price pair is "approximately" primal and dual optimal if the ϵ-CS condtions are satisfied. This intuition was verified in [BHT87], where it was shown that if a feasible flow-price vector pair (x, p) satisfies ϵ-CS, then x and p are primal and dual optimal, respectively, within a factor that is proportional to ϵ (see the following Prop. 6). Thus we may develop methods that operate by satisfying the ϵ-CS conditions and get as close to optimality as desired by making ϵ small enough.

In order to derive such methods, we first need to define a simple mechanism to move around the ϵ-CS diagram of Fig. 2 while changing one price at a time, and working towards primal feasibility. Various mechanisms can effect such price changes and Fig. 3 illustrates some of the possibilities. In particular, starting from a point on the characteristic curve of arc (i, j), we can follow any direction around that point and change the price p_i or the price p_j, and/or the flow x_{ij} simultaneously until $(x_{ij}, p_i - p_j)$ is either on the characteristic curve or is within a distance of ϵ above or below the characteristic curve of arc (i, j). For example, if node i has positive surplus, by increasing the flow of an outgoing arc (i, j) or by decreasing the flow of an incoming arc (j, i), the surplus of i will be decreased, while the surplus of j will be increased by an equal amount. This is the basic mechanism for moving flow from nodes of positive surplus to nodes of negative surplus, thus working towards primal feasibility. It is possible, however, that node i has positive surplus, while the flow of none of the outgoing arcs (i, j) can be increased and the flow of none of the incoming arcs (j, i) can be decreased without violating the ϵ-CS conditions. In this case the method increases the price of node i in order to "make room" in the ϵ-CS diagram for a subsequent flow change.

Our goal in this paper is to develop auction methods solving the convex cost network flow problem, which are characterized by price changes that are decoupled from the flow changes. The development of the algorithmic framework for auction methods will proceed in four steps:

Step 1: We will define a naive algorithm performing two basic operations: price rises on the nodes

Figure 3: Starting from any point on the characteristic curve (dark points) of arc (i, j), a new point on the characteristic curve can be obtained in a variety of ways. The figure depicts a few such examples where the flow and price differential for arc (i, j) are changed simultaneously according to some linear relation.

and flow pushes on the arcs while maintaining the $\epsilon-$CS conditions. The algorithm is naive because to guarantee convergence, we will need to further specify how the price rises and flow pushes should be effected.

Step 2: We will prove that if each price rise is bounded below, the naive algorithm can perform a finite number of price rises.

Step 3: We will define how flow pushes can be effected on arcs so as to ensure a lower bound on each price rise.

Step 4: We will discuss a key property needed in order to bound the number of flow pushes that can be performed by the naive algorithm.

In the sections that will follow, we will develop algorithms that will satisfy the properties required by the algorithmic framework and we will be able to prove their validity and estimate complexity bounds.

We proceed with the first step of our analysis and define the following two operations that can be performed on a node i with positive surplus :

(a) A *price rise* on node i, which increases the price p_i by the maximum amount that maintains ϵ-CS, while leaving all arc flows unchanged.

(b) A *flow push* (also called a δ-*flow push*) along an arc (i, j) [or along an arc (j, i)], which increases (i, j) [or decreases (j, i)] by an amount $\delta \in (0, g_i]$ that maintains ϵ-CS, while leaving all node prices unchanged.

Let us assume now that we have a naive method that at each iteration picks a node of positive surplus and performs one or more of the above operations. Let us further assume that each price rise is at least some fraction of ϵ, say $\beta\epsilon$, where $0 < \beta < 1$. Therefore, such a method has the following properties:

1. If the initial flow-price pair satisfies ϵ-CS, the method preserves ϵ-CS and the prices are monotonically nondecreasing.

2. Once the surplus of a node becomes nonnegative, it remains nonnegative for all subsequent iterations. The reason is that a flow push at a node i cannot make the surplus of i negative (cf. operation (b)) and cannot decrease the surplus of neighboring nodes.

3. If at some iteration a node has negative surplus, then its price must be equal to its initial price. This is a consequence of observation 2 above and the fact that price changes occur only at nodes with positive surplus.

4. Each price rise is at least $\beta\epsilon$, where $0 < \beta < 1$.

Based on these porperties, we can proceed to the second step of our analysis and prove a bound for the total number of price increases that the naive method can perform on any node. The proof is patterned after that for the ϵ-relaxation method for linear cost case [Ber86a], [BeE88]. A variant of the proof specialized to the convex cost ϵ-relaxation methods (and with $\beta = 0.5$) was presented in [BPT96]. We present here the proof for the naive method for completeness.

Proposition 3: Assume that for some integer $K \geq 1$, the initial price vector p^0 for the naive method satisfies $K\epsilon$-CS together with some feasible flow vector x^0. Then, the naive method performs at most $(K+1)(N-1)/\beta$ price rises per node.

Proof: Consider the pair (x, p) at the beginning of an iteration of the naive algorithm. 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 path H from t to s that contains no cycles and is such that:

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

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

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 path H from t to s that contains no cycle and 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

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

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

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

$$p_i^0 - p_j^0 \in [f_{ij}^{-}(x_{ij}^0) - K\epsilon, f_{ij}^{+}(x_{ij}^0) + K\epsilon], \qquad \forall\ (i,j) \in \mathcal{A}. \tag{10}$$

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

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

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

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

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

where $|H|$ denotes the number of arcs of the path H. We observed earlier 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 $p_t = p_t^0$. Since the path contains no cycles, we also have that $|H| \leq N-1$. Therefore, Eq. (11) yields

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

Since only nodes with positive surplus can increase their prices and, by Prop. 2, 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 result of the preceding proposition is remarkable in that the bound on the number of price changes is independent of the cost functions, but depends only on

$$K^0 = \min\{K \in \{0,1,2,\ldots\} \mid (x^0,p^0) \text{ satisfies } K\epsilon\text{-CS for some feasible flow vector } x^0\ \},$$

which is the minimum multiplicity of ϵ by which CS is violated by the starting price together with some feasible flow vector. This result will be used later to prove a particularly favorable complexity bound for the ϵ-relaxation and auction/sequential shortest path methods of later sections. 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).

The third step of our analysis defines the way a flow-push is performed. This involves defining which arcs are eligible for a flow push and the amount of flow that we will push along such arcs. The definitions that follow will also allow us to satisfy the lower bound on the amount of each price rise that we assumed for proposition 1.

For a 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 sets of arcs

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

and

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

where where β is a scalar with $0 < \beta < 1$. Figure 4 illustrates when an arc (i, j) is in the push list of i and when it is in the push list of j. We note that a definition of the push list that is used in practice is given by replacing the term $(1 - \beta)\epsilon$ with $\epsilon/2$ (see [BPT96]). The subsequent analysis applies, with minor modifications, to that case.

Figure 4: 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.

An arc (i, j) [or (j, i)] in the push list of i is said to be *unblocked* if there exists a $\delta > 0$ such that

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

[or $p_j - p_i \leq f_{ji}^-(x_{ji} - \delta)$, respectively]. For an unblocked push list arc, the supremum of δ for which the above relation holds is called the *flow margin* of the arc. The flow margin of an arc (i, j) is illustrated in Fig. 5. An important property is the following:

Proposition 2: The arcs in the push list of a node are unblocked.

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

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

and thus, based on the definition of Eq. (13a), (i,j) cannot be in the push list of node i. A similar argument proves 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.**

Figure 5: The flow margin of an unblocked push list arc.

The above definitions also allow us to prove a lower bound on each price rise increment.

Proposition 3: Assume that a price rise is performed on a positive surplus node if and only if its push list is empty. Then each price rise increment is 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 $p_i - p_j - f_{ij}^+(x_{ij}) \le (1-\beta)\epsilon$, and for every arc $(j,i) \in \mathcal{A}$ we have $p_j - p_i - f_{ji}^-(x_{ji}) \ge -(1-\beta)\epsilon$. This implies that all elements of the following sets of positive numbers:

$$S^+ = \left\{ p_j - p_i + f_{ij}^+(x_{ij}) + \epsilon \mid (i,j) \in \mathcal{A} \right\},$$

$$S^- = \left\{ p_j - p_i - f_{ji}^-(x_{ji}) + \epsilon \mid (j,i) \in \mathcal{A} \right\}$$

are greater than or equal to $\beta\epsilon$. Since a price rise at i increases p_i by the increment $\gamma = \min\{S^+ \cup S^-\}$, the result follows. **Q.E.D.**

We proceed now with the forth and final step of our development of the algorithmic framework, where we identify a key property required to bound the total number of flow pushes that can be performed. For a given pair (x,p) satisfying ϵ-CS, consider an 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 $L^-(i)$ 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* $G^* = (\mathcal{N}, \mathcal{A}^*)$. Note that an arc can be in the push list of at most one node, so the admissible graph is well defined.

We must now choose an initial flow-price vector pair (x, p) satisfying ϵ-CS, and such that the corresponding admissible graph G^* is acyclic. Acyclicity of the admissible graph is important in order to bound the number of flow pushes that a method would perform, as we shall prove in later sections. One possibility is to select an initial price vector p^0 and to set the initial arc flow x^0_{ij} for every arc $(i, j) \in \mathcal{A}$ so that the flow-price pair (x^0, p^0) satisfies 0-CS; that is

$$f^-_{ij}(\xi^0) \leq p^0_i - p^0_j \leq f^+_{ij}(\xi^0), \qquad \forall\ (i, j) \in \mathcal{A}. \tag{14}$$

It can be seen that with this choice, ϵ-CS is satisfied for every arc $(i, j) \in \mathcal{A}$, and that the initial admissible graph is empty and thus acyclic.

We now have all the necessary ingredients to develop auction methods that solve the convex cost network flow problem. In particular, we have defined the operations performed and the initialization of such methods and we have proved properties that will insure the validity and termination of these methods. We only need to define the specific order that we will effect price rises and flow pushes. In the next sections we will define and prove the validity of two such methods the ϵ-relaxation and the auction/sequential shortest path method. We will also present surprisingly favorable complexity results.

3. THE ϵ-RELAXATION METHOD

The first auction method that we extend to the convex cost network flow problem is the ϵ-relaxation method. The method has also been presented in [BPT96] but we derive it here from the general algorithmic framework.

The iteration is as follows.

Typical Iteration of the ϵ-Relaxation Method

Step 1: Select a node i with positive surplus g_i; 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, where

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

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.

We make the following observations about the ϵ-relaxation method, similar to the ones we made for the naive method of Section 2:

1. The method preserves ϵ-CS and the prices are monotonically nondecreasing. This is because the initial flow-price pair satisfies ϵ-CS, and Steps 2 and 3 of the method preserve ϵ-CS.

2. Once the surplus of a node becomes nonnegative, it remains nonnegative for all subsequent iterations. The reason is that a flow push at a node i cannot make the surplus of i negative (cf. Step 2), and cannot decrease the surplus of neighboring nodes.

3. If at some iteration a node has negative surplus, then its price must be equal to its initial price. This is a consequence of observation 2 above and the fact that price changes occur only at nodes with positive surplus.

4. Each price rise increment is at least $\beta\epsilon$. This is a consequence of Step 3 and Proposition 3.

To prove the termination of the ϵ-relaxation method, we first prove that the total number of price rises that the method can perform is bounded. In particular, Proposition 1 applies and we conclude that the ϵ-relaxation method can perform at most $(K+1)(N-1)/\beta$ price rises per node.

In order to show that the number of flow pushes that can be performed between successive price increases is finite, we first prove that the method maintains the acyclicity of the admissible graph (see also [BPT96]).

Proposition 4: The admissible graph remains acyclic throughout the ϵ-relaxation method.

Proof: We use induction. Initially, the admissible graph G^* is empty, so it is trivially acyclic. 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 at a node i, no cycle involving i is created. We note that, after a price rise at 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 at 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. (5), (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 increase at i, node i cannot have any incoming incident 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 from i to j exists in G^*. Node j is then called a *successor* of i. Observe that flow is pushed towards the successors of a node and since G^* is acyclic, flow cannot be pushed from a node to any of its predecessors. A δ-flow push along an arc in G^* is said to be *saturating* if δ is equal to the flow margin of the arc. By our choice of δ (see Step 2 of the method), a nonsaturating flow push always exhausts (i.e., sets to zero) the surplus of the starting node of the arc. Thus we have the following proposition.

Proposition 5: The number of flow pushes between two successive price increases (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 increases 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 $\{i^k\}$. 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. 5, we can further show that the number of flow pushes between successive price increases is at most $(N+1)A$, from which a complexity result for the ϵ-relaxation method may be derived. However, an implementation of the method with a sharper complexity bound has been presented in [BPT96]. We will review the implementation and the corresponding results later this section.

Propositions 1 and 5 prove that the ϵ-relaxation method terminates. Upon termination, we have that the flow-price vector pair satisfies ϵ-CS and that the flow vector is feasible since the surplus of all nodes will be zero. The following proposition, due to [BHT87], shows that the flow vector and the price vector obtained upon termination are primal optimal and dual optimal within a factor that is essentially proportional to ϵ.

Proposition 6: 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

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

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

Proposition 6 does not give an estimate of how small ϵ has to be in order to achieve a certain degree of optimality. However, in the common case where finiteness of the arc cost functions f_{ij} imply lower and upper bounds on the arc flows, Prop. 6 together with the fact $q\big(p(\epsilon)\big) \ge -f^*$ yields such an estimate for ϵ (see [BPT96]).

Complexity Analysis

We now derive a bound on the running time of the ϵ-relaxation method. The analysis was orginally presented in the Ph.D. thesis of the second author [Pol95] and in the subsequent publication [BPT96]. We review here the basic ideas and provide the main results of the analysis for completeness. We will also use these results for the analysis of the auction/sequential shortes paths algorithm of the next section. The reader is referred to the above publications for a more detailed treatment of the compelxity analyis of the ϵ-relaxation algorithm.

Since the cost functions are convex, it is not possible to express the size of the problem in terms of the problem data. To deal with this difficulty, we introduce a set of simple operations performed by the 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. (6). We will thus estimate the total number of simple operations performed by the method (see the following Prop. 8).

To obtain a sharper complexity bound, we introduce an order in which the nodes are chosen in Step 1 of each iteration. This rule is based on the *sweep implementation* of the ϵ-relaxation method, which was introduced in [Ber86a] and was analyzed in more detail in [BeE88], [BeT89], and [BC91] for the linear cost network flow problem. 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. (6), 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 a node on which we perform an ϵ-relaxation iteration, and let N_i be the subset of nodes of T that are after i in T. If the price of i changes, then node i is removed from its position in T and placed in the first position of T. The next node chosen for 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 picked. It can be shown (see the references cited earlier) that with this rule of repositioning nodes following a price change, the list order is consistent with the successor order implied by the admissible graph throughout the method.

A *sweep cycle* is a set of iterations whereby all nodes are chosen once from the list T and an ϵ-relaxation iteration is performed on those nodes that have positive surplus. 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 complexity analysis follows the line of the corresponding analysis for the linear cost problem. We will review here only the main results. The reader is referred to [BPT96] for the proofs. First we have a proposition that estimates the number of sweep cycles required for termination.

Proposition 7: Assume that for some integer $K \geq 1$, the initial price vector p^0 for the sweep implementation of the ϵ-relaxation method satisfies $K\epsilon$-CS together with some feasible flow vector x^0. Then, the number of sweep cycles up to termination is $O(KN^2)$.

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 increases.

(2) The computation required for saturating δ-flow pushes.

(3) The computation required for nonsaturating δ-flow pushes.

Proposition 8: Assume that for some $K \geq 1$ the initial price vector p^0 for the sweep implementation of the ϵ-relaxation method satisfies $K\epsilon$-CS together with some feasible flow vector x^0. Then, the method requires $O(KN^3)$ operations up to termination.

It is well known that the theoretical and the practical performance of the ϵ-relaxation method can be improved by scaling. A scaling approach in connection with the ϵ-relaxation method for linear cost problems, is *ϵ-scaling.* This approach was originally introduced in [Ber79] as a means of improving the performance of the auction algorithm for the assignment problem. Its complexity analysis was given in [Gol87] and [GoT90].

The key idea of ϵ-scaling is to apply the ϵ-relaxation method several times, starting with a large value of ϵ and to successively reduce ϵ up to a final value that will give the desirable degree of accuracy to our solution. Furthermore, the price and flow information from one application of the method is transferred to the next. The ϵ-scaling implementation of the ϵ-relaxation method is presented in detail in [BPT96]. The result is summarized in the following proposition, where $\overline{\epsilon}$ is a desirable value for ϵ on termination and ϵ^0 is sufficiently large so that the initial price vector p^0 satisfies ϵ^0-CS with some feasible flow vector x^0.

Proposition 9: The running time of the ϵ-relaxation method using the sweep implementation and ϵ-scaling as described above is $O\big(N^3 \ln(\epsilon^0/\overline{\epsilon})\big)$ operations.

We note that a complexity bound of $O\big(NA\ln(N)\ln(\epsilon^0/\overline{\epsilon})\big)$ 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. We finally note that obtaining sharper complexity estimates for our method when applied to special classes of problems, such as those involving quadratic arc cost functions, remains an interesting subject for further research.

4. THE AUCTION/SEQUENTIAL SHORTEST PATH (ASSP) METHOD

The auction/sequential shortest path (ASSP) algorithm was proposed in [Ber92] in the context of the classical minimum cost flow problem where the arc costs are linear. In this section we will extend it to the case of convex costs, based on the general algorithmic framework of Section 2 and the analysis for the ϵ-relaxation method of Section 3.

The ϵ-relaxation algorithm analyzed in the previous sections performed flow pushes and price increases on positive surplus nodes using local information about their incident arcs. The ASSP algorithm operates in a different way; it does not make any flow push until an unblocked path of minimum cost connecting a source and a sink has been found. Once such an unblocked path is found, a flow is pushed along the path from the source to the sink. Thus after such a flow push the surplus of a source decreases and the deficit of a sink decreases, whereas the surpluses of the rest of the nodes in the graph remain unchanged. In contrast the ϵ-relaxation algorithm allowed a node to decrease its own surplus by increasing the surplus of a neighboring node.

We introduce some concepts that are essential for the ASSP algorithm, specifically, the notion of a path and the operations performed on it. A path P is a sequence of nodes $(n_1, n_2, \ldots, n_k)$ and a corresponding sequence of $k-1$ arcs such that the ith arc on 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). We denote by $s(P)$ and $t(P)$ the starting node and the terminal node respectively of path P. We also define the sets P^+, P^- containing the forward and reverse arcs of P, respectively. A path that consists of a single node (and no arcs) is called a *degenerate* path. A path is called *simple* if it has no repeated nodes. It is called *unblocked* if all of its arcs are unblocked. An unblocked path starting at a source and ending at a sink is called an *augmenting path.* An *augmentation* (compare with the definition of a *flow push* we gave in Section 2) along such a path starting at a source $s(P)$ and ending at a sink $t(P)$, consists of increasing the flow of all the arcs in P^+, decreasing the flow of all arcs in P^-, decreasing the surplus of the source $g_{s(P)}$, and increasing the surplus of the sink $g_{t(P)}$ by the same amount

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

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 corresponding terminal arc. 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 $(n_1, n_2, \ldots, n_k, n_{k+1})$ and adds to P the corresponding arc.

The ASSP algorithm, described formally below, maintains a flow-price pair satisfying ϵ-CS and also a simple path P, starting at some positive surplus node. At each iteration, the path P is either extended or contracted. In case of a contraction the price of the terminal node of P is strictly increased. In the case of an extension, no price change occurs, but if the new terminal node has negative surplus then an augmentation along P is performed. Following an augmentation, P is replaced by the degenerate path that consists of a single node with positive surplus and the process is repeated. The algorithm terminates when all nodes have nonnegative surplus. Then either all nodes have zero surplus and the flow vector x is feasible, or else some node has negative surplus showing that the problem is infeasible. The algorithm is initialized in the same way as the ϵ-relaxation algorithm, rendering the initial admissible graph acyclic. As we discussed in Section 2, for these choices ϵ-CS is satisfied and the initial admissible graph is acyclic since its arc set is empty. A typical iteration of the algorithm is as follows:

Typical Iteration of the ASSP Method

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

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):** Pick an arc (i, j) (or (j, i)) from the push list of i and extend P. 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. Go to Step 1.

We proceed now to establish the basic properties that will allow us to prove the validity of the algorithm. The analysis will proceed in similar steps as for the naive algorithm of Section 2. We first prove the following proposition:

Proposition 10: Suppose that at the start of an iteration:

1. (x, p) satisfies ϵ-CS and the corresponding admissible graph is acyclic.

2. P belongs to the admissible graph.

   Then the same is true at the start of the next iteration.

Proof: Suppose that the iteration involves a contraction. Then, by definition, the price rise preserves the ϵ-CS conditions. Since only the price of i changed and no arc flow changed, the admissible graph remains unchanged except for the incident arcs to i. In particular, all incident arcs of i in the admissible graph at the beginning of the iteration are deleted and the arcs in the push list of i at the end of the iteration are added. Since all these arcs are outgoing from i in the admissible graph, a cycle cannot be formed. Finally, after the contraction, P does not contain the terminal node i, so it belongs to the admissible graph before the iteration. Thus P consists of arcs that still belong to the admissible graph after the iteration.

Suppose now that the iteration involves an extension. Since the extension arc (i, j) or (j, i) is unblocked and belongs to the push list of i, it belongs to the admissible graph and thus P belongs to the admissible graph after the extension. Since no flow or price changes with an extension, the ϵ-CS conditions and the admissible graph do not change after an extension. If there is a subsequent augmentation because of Step 5, the ϵ-CS conditions are not affected, while the admissible graph will not gain any new arcs, so it remains acyclic. **Q.E.D.**

The above proof also demonstrates the importance of relaxing the CS conditions by ϵ. If we had taken $\epsilon = 0$ then the preceding proof would break down, and the admissible graph might not remain acyclic after an augmentation. For example, if following an augmentation, the flow of some arc (i, j) with linear cost and flow margin δ_{ij} lies strictly between the flow x_{ij} it had before the augmentation and $x_{ij} + \delta_{ij}$, then both the arc (i, j) and the arc (j, i) would belong to the admissible graph closing a cycle.

We make the following observations about the ASSP method, similar to the ones we made for the naive method of Section 2:

1. The method preserves ϵ-CS and the prices are monotonically nondecreasing.

2. Once the surplus of a node becomes nonnegative, it remains nonnegative for all subsequent iterations. The reason is that an augmentation along a path P cannot make the surplus of the source $s(P)$ negative, cannot decrease the surplus of terminal node $t(P)$ and does not change the surplus of the intermediate nodes of the path P (cf. Step 5).

3. If at some iteration a node has negative surplus, then its price must be equal to its initial

price. This is a consequence of observation 2 above and the fact that price changes occur only at nodes with positive surplus (cf. Step 5, where if a node of negative surplus is encountered, the algorithm performs an augmentation).

4. Each price rise increment is at least $\beta\epsilon$. This is a consequence of Step 3 and Proposition 3.

We conclude that Proposition 1 applies to our algorithm. Thus the number of price rises that can be performed by our algorithm is bounded. In particular, if the initial price vector p^0 for the ASSP algorithm satisfies $K\epsilon$-CS with some feasible flow vector x^0 for some $K \geq 1$, then the algorithm performs $O(KN)$ price increases per node of positive surplus.

We will now complete the proof of the validity of the algorithm. First, we ensure that a path from a source to some sink can be found. The sequence of iterations between successive augmentations (or the sequence of iterations up to the first augmentation) will be called an *augmentation cycle.* Let us fix an augmentation cycle and let $\hat{p}$ be the price vector at the start of this cycle. 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 $f_{ij}^{+}(x_{ij}) + \hat{p}_j - \hat{p}_i + \epsilon$ and an arc (j,i) with length $\hat{p}_i - f_{ij}^{-}(x_{ij}) - \hat{p}_j + \epsilon$. 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 (x,p) satisfy ϵ-CS the arc lengths of the reduced graph are nonnegative. Furthermore, the reduced graph contains no zero length cycles since such a cycle would belong to the admissible graph which we proved to be acyclic. During an augmentation cycle the reduced graph remains unchanged, since no arc flow changes except for the augmentation at the end.

It can now be seen that the augmentation cycle is just the auction shortest path algorithm of [Ber91] and it constructs a shortest path in the reduced graph $G_{\mathcal{R}}$ starting at a source $s(P)$ and ending at a sink $t(P)$. By the theory of the auction shortest path algorithm, a shortest path in the reduced graph from a source to some sink will be found if one exists. Such a path will exist if the convex cost flow problem is feasible.

Finally, we have to bound the total number of augmentations the algorithm performs between two successive price rises (not necessarily at the same node). First we observe that there can be at most N successive extensions before either a price rise occurs or an augmentation is possible. We say that an augmentation is *saturating* if the flow increment is equal to the flow margin of at least one arc of the path. The augmentation is called *exhaustive* if, after it is performed, the surplus of either the starting node $s(P)$ or the terminal $t(P)$ becomes zero (compare with similar definitions we gave in Section 3). An augmentation cannot introduce new arcs in the admissible graph G^* since a saturating augmentation along a path P removes from G^* all the arcs of P whose flow margin is equal to the flow increment. Thus there can be at most

$O(A)$ saturating augmentations. Furthermore, if a node has zero surplus at some point in the algorithm, then its surplus remains zero for all subsequent iterations of the algorithm. Thus there can be at most $O(N)$ exhaustive flow pushes. Therefore at most $O(NA)$ augmentations can be performed between successive prices rises. Thus the algorithm terminates with a feasible flow vector, and a flow-price pair satisfying the ϵ-CS conditions.

Complexity Analysis

The ASSP algorithm, as described above, has a complexity similar to the ϵ-relaxation without the sweep implementation. To get a better complexity bound for the algorithm we could modify the way it operates. For example, we consider a hybrid algorithm which performs ϵ-relaxation iterations according to the sweep implementation along with some additional ASSP iterations. This hybrid algorithm points out the issues that may lead to an efficient implementation of the ASSP algorithm. In particular, all the nodes are kept in a linked list T which is traversed from the first to the last element. The initial list is arbitrary. During the course of the algorithm, the list is updated as follows: Whenever a price rise occurs at a node, the node is removed from its current position at the list and is placed at the first position of the list. Furthermore, we perform a fixed number C of ASSP iterations at the end of which we perform an ϵ-relaxation iteration at $s(P)$ and pick a new source from T. Let $N_{s(P)}$ contain the nodes of T that are lower than $s(P)$ in T. The next node to be picked for an iteration is the node $i' \in N_{s(P)}$ with positive surplus which ranks highest in T, if such a node exists. Otherwise, we declare that a cycle of the algorithm has been completed and the positive surplus node ranking highest in T is picked.

This implementation of ASSP is in essence the sweep implementation of the ϵ-relaxation algorithm with some additional (at most C at a time) iterations of the ASSP algorithm. In particular, we note that a cycle for the hybrid algorithm, is the set of iterations whereby all nodes of positive surplus in T performed either an ϵ-relaxation iteration or at least one a price rise. Whenever a price increase occurs on a node i, then i can have no predecessors and is placed at the top of the list. Thus, the nodes appear in the list before all their predecessors. Furthermore, if no price rise has occurred during a cycle, then all nodes with positive surplus have performed an ϵ-relaxation iteration and Proposition 8 applies. We conclude that from the analysis of Section 3 that the hybrid algorithm has a similar complexity bound with the ϵ-relaxation algorithm with the sweep implementation. We can further make an ϵ-scaling implementation and obtain a result similar to Proposition 9.

The above implementation is of importance in practice since it improves the performance of the algorithm. In particular, if the ASSP algorithm has made numerous iterations with some starting node $s(P)$ without finding a path to a sink, we may deduce that the path to some sink is "long" and many more iterations may be needed for the algorithm to find it. We can make an augmentation along our current path and pick a new node on which to iterate. Thus we may benefit in two ways. First, if the terminal node of the path is not a source, then, after the augmentation, we have a source potentially closer to a sink than $s(P)$. Secondly, we allow the algorithm to pick a new source on which to iterate, enabling sources that are close to sinks to find unblocked paths first. Experimentation has shown ([BeP94]) that such an implementation is successful in the case of linear costs, especially when the paths from sources to sinks are unusually long.

5 COMPUTATIONAL RESULTS

We have developed and tested two experimental Fortran codes implementing the methods of this paper for convex cost problems. We have chosen $\beta = 0.5$ for simplicity in the implemetnation. The first code, named NE-RELAX-F, implements the ϵ-relaxation method with the sweep implementation and ϵ-scaling as described in Section 3. The second code, named ASSP-NE implements the auction/sequential shortest path method with the implementation enhancements described in Section 4. 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 the codes for convex cost problems. In particular, all computations are done in real rather than integer arithmetic, and ϵ-scaling, rather than arc cost scaling, is used. Furthermore, the updating of the push lists and prices were modified to improve efficiency. Otherwise, the sweep implementation and the general structure of the codes for linear and convex cost problems are identical.

The codes NE-RELAX-F and ASSP-NE were compared to two existing Fortran codes NRELAX and MNRELAX from [BHT87]. The latter implement the relaxation method for, respectively, strictly convex cost and convex 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 terminated according to the same criterion, namely that the cost of the feasible flow vector and the cost of the price vector must agree in their first 12 digits.

For our testing, we used convex linear/quadratic problems corresponding to the case of (P) where

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

for some a_{ij}, b_{ij}, and c_{ij} with $-\infty < a_{ij} < \infty$, $b_{ij} \ge 0$, and $c_{ij} \ge 0$. 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, 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. 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 and auction methods 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 and auction methods to problem ill-conditioning.

Ill-conditioned problems were created by assigning to some of the arcs have 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 well-conditioned strictly convex quadratic cost problems generated using CHAINGEN (Table 2); the third set comprises ill-conditioned strictly convex quadratic cost problems and mixed linear/quadratic cost problems generated using NETGEN (Table 3). The running time of the codes on these problems are shown in the last three to four columns of Tables 1–3. In all problems, the ϵ-relaxation and action codes were run to the point where they yielded higher or comparable solution accuracy than the relaxation codes. From the running times we can draw the following conclusions: First, the ϵ-relaxation code NE-RELAX-F and ASSP-NE have similar performance and both 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. The only explanation we have for this phenomenon is the favorable complexity analysis that we presented in this paper.

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.

[Ber85] Bertsekas, D. P., "A Unified Framework for Minimum Cost Network Flow Problems," Mathematical Programming, Vol. 32, 1985, pp. 125-145.

[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.

[BC91] Bertsekas, D. P., Castanon, D. A. "A Generic Auction Algorithm for the Minimum Cost Network Flow Problem," Laboratory for Information and Decision Systems Report LIDS-P-2084, M.I.T., Cambridge, MA, 1991, Compuatational Optimization and Applications, 1994.

[BCE95] Bertsekas, D. P., Castanon, D., Eckstein, J., and Zenios, S. A., "Parallel network optimization survey", to appear in Encyclopedia of OR.

[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.

[BPT96] Bertsekas, D. P., Polymenakos, L. C., and Tseng, P., "An ϵ-Relaxation Method for Separable Convex Cost Network Flow Problems," accepted for publication, SIAM Journal on Optimization.

[BeP94] Polymenakos, L. C., Personal Communication with Bertsekas, D. P. on computational experimentations for the linear cost network flow problem.

[BeT88] Bertsekas, D. P., and Tseng, P., "Relaxation Methods for Minimum Cost Ordinary and Generalized Network Flow Problems," Operations Research, Vol. 36, 1988, pp. 93-114.

[BeT94] Bertsekas, D. P., and Tseng, P., " RELAX-IV: A Faster Version of the RELAX Code for Solving Minimum Cost Flow Problems," Laboratory for Information and Decision Systems Report P-2276, M.I.T., Cambridge, MA, 1994.

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

[BlJ92] Bland, R. G., and Jensen, D. L., "On the Computational Behavior of a Polynomial-Time Network Flow Algorithm," Mathematical Programming, Vol. 54, 1992, pp. 1-39.

[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.

[EdK72] Edmonds, J., and Karp, R. M., "Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems," Journal of the ACM, Vol. 19, 1972, pp. 248-264.

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

[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.

[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.

[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.

[LiZ91] Li, X., and Zenios, S. A., "Data Parallel Solutions of Min-Cost Network Flow Problems Using ϵ-Relaxations," Department of Decision Sciences Report 1991-05-20, University of Pennsylvania, Philadelphia, PA, 1991.

[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.

Optimization:

[Pol94] Polymenakos, L. C. "Parallel Shortest Path Auction Algorithms," Parallel Computing, Vol. 20, 1994, pp. 1221-1247.

[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.

[Roc80] Röck, H., "Scaling Techniques for Minimal Cost Network Flows," in Discrete Structures and Algorithms, Edited by U. Pape, Carl Hanser, München, Germany, 1980, pp. 181-191.

[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.

[Tse86] Tseng, P., "Relaxation Methods for Monotropic Programming Problems," Operations Research Center Ph.D. Thesis, M.I.T., Cambridge, MA, 1986.

[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.