December 1991 LIDS-P-2084

A GENERIC AUCTION ALGORITHM FOR THE MINIMUM COST NETWORK FLOW PROBLEM

by

Dimitri P. Bertsekas[1] and David A. Castanon[2]

Abstract

In this paper we broadly generalize the assignment auction algorithm to solve linear minimum cost network flow problems. We introduce a generic algorithm, which contains as special cases a number of interesting algorithms, including the ϵ-relaxation method, the auction algorithm for transportation problems, a new network auction algorithm, and a new algorithm for the k node-disjoint shortest path problem. We provide a broadly applicable complexity analysis of the generic algorithm, and we demonstrate the performance of various special cases of the algorithm via computational experimentation.

[1] Department of Electrical Engineering and Computer Science, M.I.T., Cambridge, Mass., 02139.
[2] Department of Electrical and Computer Engineering, Boston University, Boston, Mass.

1. INTRODUCTION

In this paper we discuss algorithms for solution of the classical minimum cost network flow problem, involving a directed graph with node set $\mathcal{N}$ and arc set $\mathcal{A}$. Each arc (i,j) has a cost coefficient a_{ij}. Letting x_{ij} be the flow of the arc (i,j), the problem is

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

subject to

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

$$b_{ij} \le x_{ij} \le c_{ij}, \qquad \forall\, (i,j) \in \mathcal{A}, \tag{2}$$

where a_{ij}, b_{ij}, c_{ij}, and s_i are given integers.

We denote by x the vector with elements x_{ij}, $(i,j) \in \mathcal{A}$. We refer to b_{ij} and c_{ij}, and the interval $[b_{ij}, c_{ij}]$ as the *flow bounds* and the *feasible flow range* of arc (i,j), respectively. We refer to s_i as the *supply* of node i.

We refer to the constraints (1) and (2) as the *conservation of flow constraints* and the *capacity constraints* respectively. A flow vector satisfying both of these constraints is called *feasible*, and if it satisfies just the capacity constraints, it is called *capacity-feasible*. If there exists at least one feasible flow vector, problem (LNF) is called *feasible* and otherwise it is called *infeasible*.

For a given flow vector x, the *divergence* of node i is defined to be the total flow coming out of i minus the total flow coming into i,

$$y_i = \sum_{\{j|(i,j)\in\mathcal{A}\}} x_{ij} - \sum_{\{j|(j,i)\in\mathcal{A}\}} x_{ji}.$$

The *surplus* of node i is defined as the difference between the supply and the divergence of i,

$$g_i = s_i - y_i; \tag{3}$$

it provides a measure of discrepancy between the specified flow into the node and the actual net flow out of the node. Note that a flow vector is feasible if and only if it is capacity-feasible and the surplus of each node is zero.

We assume that there exists at most one arc in each direction between any pair of nodes, but this assumption is made for notational convenience and can be easily dispensed with. We denote the numbers of nodes and arcs by N and A, respectively. We also denote by C the maximum absolute value of the cost coefficients,

$$C = \max_{(i,j)\in\mathcal{A}} |a_{ij}|.$$

We formulate a well-known dual problem to (LNF) by associating a Lagrange multiplier p_i (also called the *price of node i*) with the ith conservation of flow constraint (1). Letting p be the vector with elements p_i, $i \in \mathcal{N}$, we can write the corresponding Lagrangian function as

$$L(x,p) = \sum_{(i,j)\in\mathcal{A}} (a_{ij} + p_j - p_i)x_{ij} + \sum_{i\in\mathcal{N}} s_i p_i. \tag{4}$$

One obtains the dual function value $q(p)$ at a vector p by minimizing $L(x,p)$ over all capacity-feasible flows x. This leads to the dual problem

$$\begin{aligned} &\text{maximize } q(p) \\ &\text{subject to no constraint on } p, \end{aligned} \tag{5}$$

with the dual functional q given by

$$q(p) = \min_x \{L(x,p) \mid b_{ij} \le x_{ij} \le c_{ij}, (i,j) \in \mathcal{A}\} = \sum_{(i,j)\in\mathcal{A}} q_{ij}(p_i - p_j) + \sum_{i\in\mathcal{N}} s_i p_i, \tag{6a}$$

where

$$q_{ij}(p_i - p_j) = \min_{x_{ij}} \{(a_{ij} + p_j - p_i)x_{ij} \mid b_{ij} \le x_{ij} \le c_{ij}\}. \tag{6b}$$

We henceforth refer to (LNF) as the *primal problem*, and note that standard duality results imply that the optimal primal cost equals the optimal dual cost. A flow-price vector pair (x,p) is said to satisfy the *complementary slackness* conditions if x is capacity-feasible and

$$x_{ij} < c_{ij} \quad \Rightarrow \quad p_i - p_j \le a_{ij}, \qquad \forall\ (i,j) \in \mathcal{A}, \tag{7a}$$

$$b_{ij} < x_{ij} \quad \Rightarrow \quad p_i - p_j \ge a_{ij}, \qquad \forall\ (i,j) \in \mathcal{A}. \tag{7b}$$

For a pair (x,p), complementary slackness and primal feasibility ($g_i = 0$ for all $i \in \mathcal{N}$) are necessary and sufficient conditions for x to be primal-optimal and p to be dual-optimal.

The special structure of the dual cost (6) motivates solution by Gauss-Seidel relaxation (or coordinate ascent methods). Relaxation methods of this type have been developed in the last few years, and have led to remarkably successful computer codes [Ber82], [BeT85], [BeT88]. The idea is to choose a single node i and change its price p_i in a direction of improvement of the dual cost, while keeping the other prices unchanged. Unfortunately there is a fundamental problem; the dual cost q is nondifferentiable (piecewise linear), and the relaxation idea may encounter difficulty at some "corner points," where the dual cost cannot be improved by changing any single node price.

The difficulty was overcome in the relaxation method proposed in [Ber82] by occasionally allowing simultaneous price changes of several nodes. An alternative approach was introduced with the *auction algorithm* for the assignment problem, which was first proposed in [Ber79]. An extension of

the auction algorithm for the minimum cost flow problem (LNF), called *ϵ-relaxation*, was first proposed in [Ber86a] and [Ber86b], and is actually mathematically equivalent to the auction algorithm [BeE88]. We refer to [Ber88], [BeE88], [BeT89], [Ber90], [GoT90], and [Ber91a] for detailed analyses of these algorithms, and to [PhZ88], [BeC89b], [BeC89c], [KKZ89], [WeZ90], [Zak90], [WeZ91], and [LiZ91] for computational results and implementations in a variety of parallel machines.

The main idea in the ϵ-relaxation method is to allow a single price p_i to change even if this worsens the dual cost. When p_i is changed, however, it is set to within a given $\epsilon > 0$ of the price that maximizes the dual cost along the ith coordinate. For ϵ small enough, it can be shown that the algorithm approaches the optimal dual cost sufficiently accurately to yield a primal-optimal solution. The ϵ-approximation mechanism here is similar to the one of the ϵ-subgradient method for nondifferentiable optimization [BeM73].

While in the ϵ-relaxation algorithm there is at most one node price change per iteration, in the auction algorithm there may be two node price changes. In particular, the price of an unassigned person is raised implicitly through a "bid" as this person is assigned to a "preferred" object (see [Ber88], and [BeT89], Section 5.3), and then the price of this object is also raised. Raising the price of the person and the preferred object simultaneously is an important feature that, we believe, accounts for the practical effectiveness of the auction algorithm. Experiments show that the ϵ-relaxation method applied to the assignment problem, is on the average far slower than the auction algorithm. This is true even when the two methods are implemented so that their worst-case complexity bound is the same, and illustrates once more the fallacy of evaluating algorithms and implementations strictly on the basis of a worst-case complexity analysis.

In this paper, we introduce a general algorithm, which contains the auction algorithm and the ϵ-relaxation method as special cases, and encompasses the idea of combining a price increase of a node with price increases of several neighboring nodes. This general algorithm can form the basis for a broad variety of algorithms tailored to the structure of particular problems. As special cases, we develop a new algorithm for general minimum cost flow problem (LNF), called *network auction*, and a new and intuitively appealing algorithm for the k node-disjoint shortest path problem.

In addition, to developing the general algorithm and establishing its termination properties, we also analyze its worst-case complexity. In its unscaled form, the general algorithm is hampered by the *price war phenomenon*, which manifests itself as protracted sequences of small price increases by a number of nodes that push flow back and forth among themselves. This phenomenon was observed experimentally and was supported by a pseudopolynomial complexity analysis when the auction algorithm was first proposed [Ber79]. As a remedy, the ϵ-scaling technique was suggested in [Ber79]. Subsequent research, starting with [Gol87], has established that with ϵ-scaling or other forms of

scaling, the auction and ϵ-relaxation algorithms have particularly favorable polynomial worst-case complexity [BeE87], [BeE88], [BeT89], [GoT90]. In this paper we use cost scaling and we show that a major special case of the general algorithm, the network auction algorithm, has an $O(N^3 \log(NC))$ running time; the data structures used here are simple and are based on the *sweep implementation* of [Ber86a] and [BeE88]. By specializing this result, we obtain an $O(N^3 \log(NC))$ running time for a suitable implementation of our earlier auction algorithm for transportation problems [BeC89a]; no polynomial complexity analysis was available for this transportation algorithm. Under the assumption that the feasible flow range of all arcs is $[0, 1]$, we show that the general algorithm has an $O(NA \log(NC))$ running time, where A is the number of arcs. This bound applies in particular to the assignment auction algorithm and to the new k node-disjoint shortest path algorithm. Finally, we show that a straightforward unsophisticated implementation of the general algorithm has a $O(N^4 \log(NC))$ running time.

The paper is organized as follows: in the next section we formulate the generic auction algorithm and establish its validity. In Section 3 we show that several known auction algorithms for assignment and transportation problems, the ϵ-relaxation method, and the network auction algorithm are all special cases of the generic algorithm. In Section 4, we give the auction algorithm for the k node-disjoint shortest path problem, and we show that it is a special case of the network auction algorithm and, by extension, a special case of the generic algorithm. In Section 5 contains our complexity analysis. Finally, in Section 6 we present a variety of computational results.

2. BASIC OPERATIONS AND THE GENERIC ALGORITHM

The algorithms of this paper maintain a price vector p, and a capacity-feasible flow vector x, such that x and p jointly satisfy a relaxed form of the usual complementary slackness conditions known as ϵ-complementary slackness (ϵ-CS for short). ϵ-CS was introduced in the context of the original proposal of the auction algorithm for the assignment problem [Ber79], and was generalized in [BeT85], [BHT87], [TsB87a], and [TsB87b] for other types of problems. Roughly, a flow-price vector pair satisfies ϵ-CS if it violates the complementary slackness conditions (7) by at most ϵ on each arc. In particular, we say that (x, p) satisfies ϵ-CS if x is capacity-feasible and

$$x_{ij} < c_{ij} \quad \Rightarrow \quad p_i - p_j \leq a_{ij} + \epsilon \qquad \forall\ (i,j) \in \mathcal{A}, \tag{8a}$$

$$b_{ji} < x_{ji} \quad \Rightarrow \quad p_i - p_j \leq -a_{ji} + \epsilon \qquad \forall\ (j,i) \in \mathcal{A}. \tag{8b}$$

The usefulness of ϵ-CS is due in large measure to the following proposition. A proof may be found in [Ber86a], [BeE88], [BeT89], [Ber91a]. Note that the proposition relies on our assumption that the problem data are integer.

Proposition 1: If $\epsilon < 1/N$, x is feasible, and x and p jointly satisfy ϵ-CS, then x is optimal for (LNF).

We now define some terminology and computational operations that play a significant role in our algorithms. Each of these definitions assumes that (x, p) is a flow-price vector pair satisfying ϵ-CS, and will be used only in that context.

Definition 1: An arc (i, j) is said to be ϵ^+-*unblocked* if

$$p_i = p_j + a_{ij} + \epsilon, \qquad x_{ij} < c_{ij}. \tag{9}$$

An arc (j, i) is said to be ϵ^--*unblocked* if

$$p_i = p_j - a_{ji} + \epsilon, \qquad b_{ji} < x_{ji}. \tag{10}$$

The *push list* of a node i, denoted P_i, is the (possibly empty) set of arcs (i, j) that are ϵ^+- unblocked, denoted P_i^+, and the arcs (j, i) that are ϵ^--unblocked, denoted P_i^-.

In all our algorithms, flow is allowed to increase only along ϵ^+-unblocked arcs and is allowed to decrease only along ϵ^--unblocked arcs. The next definition specifies the type of flow changes considered.

Definition 2: For an arc (i, j) [or arc (j, i)] of the push list P_i of node i, let δ be a scalar such that $0 < \delta \leq c_{ij} - x_{ij}$ ($0 < \delta \leq x_{ji} - b_{ji}$, respectively). A *$\delta$-push at node i on arc (i, j)* [(j, i), respectively] consists of increasing the flow x_{ij} by δ (decreasing the flow x_{ji} by δ, respectively), while leaving all other flows, as well as the price vector unchanged. A *saturating push* of node i on arc (i, j) [arc (j, i), respectively] is a δ-push with $\delta = c_{ij} - x_{ij}$ ($\delta = x_{ji} - b_{ji}$, respectively).

The next operation consists of raising the prices of a subset of nodes by the maximum common increment γ that will not violate ϵ-CS.

Definition 3: A *price rise* of a nonempty, strict subset of nodes I (i.e., $I \neq \emptyset$, $I \neq \mathcal{N}$), consists of leaving unchanged the flow vector x and the prices of nodes not belonging to I, and of increasing the prices of the nodes in I by the amount γ given by

$$\gamma = \begin{cases} \min\{S^+ \cup S^-\}, & \text{if } S^+ \cup S^- \neq \emptyset, \\ 0, & \text{if } S^+ \cup S^- = \emptyset, \end{cases} \tag{11}$$

where S^+ and S^- are the sets of scalars given by

$$S^+ = \{p_j + a_{ij} + \epsilon - p_i \mid (i,j) \in \mathcal{A} \text{ such that } i \in I, j \notin I, x_{ij} < c_{ij}\}, \tag{12}$$

$$S^- = \{p_j - a_{ji} + \epsilon - p_i \mid (j,i) \in A \text{ such that } i \in I, j \notin I, x_{ji} > b_{ji}\}. \tag{13}$$

In the case where the subset I consists of a single node i, a price rise of the singleton set $\{i\}$ is also referred to as a *price rise of node i*. If the price increment γ of Eq. (11) is positive, the price rise is said to be *substantive* and if $\gamma = 0$, the price rise is said to be *trivial*. [Every scalar in the sets S^+ and S^- of Eqs. (12) and (13) is nonnegative by the ϵ-CS conditions (8a) and (8b), respectively, so we have $\gamma \geq 0$. A trivial price rise changes neither the flow vector nor the price vector; it is introduced to facilitate the presentation. Note that a price rise of a single node i is substantive if and only if the set $S^+ \cup S^-$ is nonempty but the push list of i is empty.]

The generic algorithm to be described shortly consists of a sequence of δ-push, and price rise operations. The following lemma lists some properties of these operations that are important in the context of the algorithm.

Lemma 1: Let (x,p) be a flow-price vector pair satisfying ϵ-CS.

(a) The flow-price vector pair obtained following a δ-push or a price rise operation satisfies ϵ-CS.

(b) Let I be a subset of nodes with positive total surplus, that is, $\sum_{i \in I} g_i > 0$. Then if the sets of scalars S^+ and S^- of Eqs. (12) and (13) are empty, problem (LNF) is infeasible.

Proof: (a) By the definition of ϵ-CS, the flow of an ϵ^+-unblocked and an ϵ^--unblocked arc can have any value within the feasible flow range. Since a δ-push only changes the flow of an ϵ^+-unblocked or ϵ^--unblocked arc, it cannot result in violation of ϵ-CS. If p and p' are the price vectors before and after a price rise operation of a set I, respectively, we have that for all arcs (i,j) with $i \in I$, and $j \in I$ or with $i \notin I$ and $j \notin I$, the ϵ-CS condition (8) is satisfied by (x,p') since it is satisfied by (x,p) and we have $p_i - p_j = p'_i - p'_j$. For arcs (i,j) with $i \in I$, $j \notin I$ and $x_{ij} < c_{ij}$ we have, using Eqs. (11) and (12),

$$p'_i - p'_j = p_i - p_j + \gamma \leq p_i - p_j + (p_j + a_{ij} + \epsilon - p_i) = a_{ij} + \epsilon,$$

so condition (8a) is satisfied. Similarly, using Eqs. (11) and (13), it is seen that for all arcs (j,i) with $i \in I$, $j \notin I$ and $x_{ji} > b_{ji}$, condition (8b) is satisfied.

(b) Since the sets S^+ and S^- are empty,

$$x_{ij} = c_{ij}, \qquad \forall\ (i,j) \in \mathcal{A} \text{ with } i \in I, j \notin I, \tag{14}$$

$$x_{ji} = b_{ji}, \qquad \forall\ (i,j) \in \mathcal{A} \text{ with } i \in I, j \notin I. \tag{15}$$

Using the definition (3) of surplus, we have

$$0 < \sum_{i \in I} g_i = \sum_{i \in I} s_i - \sum_{\{(i,j) \in \mathcal{A} \mid i \in I, j \notin I\}} x_{ij} + \sum_{\{(j,i) \in \mathcal{A} \mid i \in I, j \notin I\}} x_{ji}, \tag{16}$$

and by combining Eqs. (14)-(16), it follows that

$$0 < \sum_{i \in I} s_i - \sum_{\{(i,j) \in \mathcal{A} \mid i \in I, j \notin I\}} c_{ij} + \sum_{\{(j,i) \in \mathcal{A} \mid i \in I, j \notin I\}} b_{ji}.$$

For a feasible vector, s_i is equal to the divergence of i, so the above relation implies that the sum of the divergences of nodes in I exceeds the capacity of the cut $[I, \mathcal{N} - I]$, which is a contradiction. Therefore, the problem is infeasible. **Q.E.D.**

The Generic Algorithm

Suppose that problem (LNF) is feasible, and consider a pair (x,p) satisfying ϵ-CS. Suppose that for some node i we have $g_i > 0$. There are two possibilities:

(a) The push list of i is nonempty, in which case a δ-push at node i is possible.

(b) The push list of i is empty, in which case the set $S^+ \cup S^-$ corresponding to the set $I = \{i\}$ [cf. Eqs. (12) and (13)] is nonempty, since the problem is feasible [cf. Lemma 1(b)]. Therefore, from Eqs. (11)-(13), a price rise of node i will be substantive.

Thus, *if $g_i > 0$ for some i and the problem is feasible, then either a δ-push or a substantive price rise is possible at node i.* Furthermore, since following a price rise at a node i, the push list of i will be nonempty [cf. Eqs. (11)-(13)], *for a feasible problem a δ-push is always possible at a node i with $g_i > 0$, possibly following a price rise at i.*

The preceding observations motivate a method, called *generic algorithm*, which uses a fixed positive value of ϵ, and starts with a pair (x,p) satisfying ϵ-CS. The algorithm terminates when $g_i \leq 0$ for all nodes i; otherwise it continues to perform iterations. Each iteration consists of a sequence of δ-pushes and price rises, including at least one δ-push, as described below.

Typical Iteration of the Generic Algorithm

Perform in sequence and in any order a finite number of δ-pushes and price rises; there should be at least one δ-push but not necessarily at least one price rise. Furthermore:

(1) Each δ-push should be performed at some node i with $g_i > 0$, and the flow increment δ must satisfy $\delta \leq g_i$.

(2) Each price rise should be performed on a set I with $g_i \geq 0$ for all $i \in I$.

The following proposition establishes the validity of the generic algorithm.

Proposition 2: Assume that the minimum cost flow problem (LNF) is feasible. If the increment δ of each δ-push is integer, then the generic algorithm terminates with a pair (x, p) satisfying ϵ-CS. The flow vector x is feasible, and is optimal if $\epsilon < 1/N$.

Proof: We first make the following observations.

(a) The algorithm preserves ϵ-CS; this is a consequence of Lemma 1.

(b) The prices of all nodes are monotonically nondecreasing during the algorithm.

(c) Once a node has nonnegative surplus, its surplus stays nonnegative thereafter. The reason is that a δ-push at a node i cannot drive the surplus of i below zero (since $\delta \leq g_i$), and cannot decrease the surplus of neighboring nodes.

(d) If at some time a node has negative surplus, its price must have never been increased up to that time, and must be equal to its initial price. This is a consequence of (c) above and of the assumption that only nodes with nonnegative surplus can be involved in a price rise.

Suppose, to arrive at a contradiction, that the algorithm does not terminate. Then, since there is at least one δ-push per iteration, an infinite number of δ-pushes must be performed at some node m on some arc (m, n) or some arc (n, m). For concreteness, assume it is arc (m, n); a similar argument applies if the arc is (n, m). Since for each δ-push, δ is integer, an infinite number of δ-pushes must also be performed at the opposite endnode n of the arc (m, n). This means that arc (m, n) becomes alternately ϵ^+-unblocked with $g_m > 0$ and ϵ^--unblocked with $g_n > 0$ an infinite number of times, which implies that p_m and p_n must increase by amounts of at least 2ϵ an infinite number of times. Thus we have $p_m \to \infty$ and $p_n \to \infty$, while either $g_m > 0$ or $g_n > 0$ at the start of an infinite number of δ-pushes.

Let $\mathcal{N}^\infty$ be the set of nodes whose prices increase to ∞; this set includes the nodes m and n. To preserve ϵ-CS, we must have, after a sufficient number of iterations,

$$x_{ij} = c_{ij} \qquad \text{for all } (i,j) \in \mathcal{A} \text{ with } i \in \mathcal{N}^\infty, j \notin \mathcal{N}^\infty, \tag{17}$$

$$x_{ji} = b_{ji} \qquad \text{for all } (j,i) \in \mathcal{A} \text{ with } i \in \mathcal{N}^\infty, j \notin \mathcal{N}^\infty. \tag{18}$$

After some iteration, by (d) above, every node in $\mathcal{N}^\infty$ must have nonnegative surplus, so the sum of surpluses of the nodes in $\mathcal{N}^\infty$ must be positive at the start of the δ-pushes where either $g_m > 0$

or $g_n > 0$. It follows using the argument of the proof of Lemma 1(b) [cf. Eqs. (14)-(16)] that

$$0 < \sum_{i \in \mathcal{N}^\infty} s_i - \sum_{\{(i,j) \in \mathcal{A} \mid i \in \mathcal{N}^\infty, j \notin \mathcal{N}^\infty\}} c_{ij} + \sum_{\{(j,i) \in \mathcal{A} \mid i \in \mathcal{N}^\infty, j \notin \mathcal{N}^\infty\}} b_{ji}.$$

For any feasible vector, the above relation implies that the sum of the divergences of nodes in $\mathcal{N}^\infty$ exceeds the capacity of the cut $[\mathcal{N}^\infty, \mathcal{N} - \mathcal{N}^\infty]$, which is impossible. It follows that there is no feasible flow vector, contradicting the hypothesis. Thus the algorithm must terminate. Since upon ternination we have $g_i \leq 0$ for all i and the problem is assumed feasible, it follows that $g_i = 0$ for all i. Hence the final flow vector x is feasible and by (a) above it satisfies ϵ-CS together with the final p. By Prop. 1, if $\epsilon < 1/N$, x is optimal. **Q.E.D.**

The example of Fig. 1 shows how the generic algorithm may never terminate even for a feasible problem, if we do not require that it performs at least one δ-push per iteration.

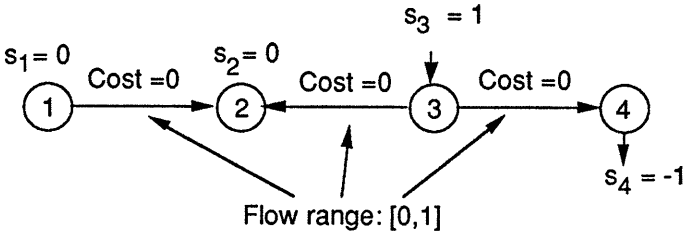

Figure 1 Example of a feasible problem where the generic algorithm does not terminate, if it does not perform at least one δ-push per iteration. Initially, all flows and prices are zero. Here, the first iteration raises the price of node 1 by ϵ. Subsequent iterations consist of a price rise of node 2 by an increment of 2ϵ followed by a price rise of node 1 by an increment of 2ϵ.

Dealing with Infeasibility

Let us consider now what happens when the problem is infeasible. Assume that the generic algorithm is operated so that for each δ-push, δ is integer. Then there are three possibilities:

(a) The algorithm terminates with $g_i \leq 0$ for all i and $g_i < 0$ for at least one i, in which case infeasibility is detected.

(b) The algorithm finds a subset of nodes I such that $\sum_{i \in I} g_i > 0$, and the sets of scalars S^+ and S^- of Eqs. (18) and (19) are empty [cf. Lemma 1(b)], in which case infeasibility is again detected.

(c) The algorithm performs an infinite number of iterations and, consequently, an infinite number of δ-pushes. In this case, from the proof of Prop. 2 it can be seen that the prices of the nodes involved in an infinite number of δ-pushes will diverge to infinity. The following proposition

gives a bound on the total price change of a node for a feasible problem. When this bound is violated, infeasibility is established.

Proposition 3: Suppose that the generic algorithm is applied to a feasible minimum cost flow problem with initial prices p_i^0. Then in the course of the algorithm, the price p_i of any node i with $g_i > 0$ satisfies

$$p_i - p_i^0 \leq (N-1)(C+\epsilon) + \max_{j \in \mathcal{N}} p_j^0 - \min_{j \in \mathcal{N}} p_j^0, \tag{19}$$

where $C = \max_{(i,j) \in \mathcal{A}} |a_{ij}|$.

Proof: Let x^0 be a feasible flow vector and let (x,p) be a flow–price vector pair generated by the algorithm prior to its termination. Suppose that $g_i > 0$ for some i. Then by using the Conformal Realization Theorem (see e.g. [Roc84], [Ber91a]) on the flow vector $x - x^0$, we conclude that there exists a node s such that $g_s < 0$, and a simple path H starting at s and ending at i such that $x_{ij} - x_{ij}^0 > 0$ for all $(i,j) \in H^+$ and $x_{ij} - x_{ij}^0 < 0$ for all $(i,j) \in H^-$, where H^+ and H^- are the sets of forward and backward arcs of H, respectively. By ϵ-CS we have

$$p_j + a_{ij} \leq p_i + \epsilon, \qquad \forall\ (i,j) \in H^+,$$

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

Adding these conditions along H, we obtain

$$p_i - p_s \leq (N-1)(C+\epsilon).$$

Since s has negative surplus, its price has not yet changed ($p_s = p_s^0$), so by subtracting p_i^0 from both sides of the above relation, we conclude that

$$p_i - p_i^0 \leq (N-1)(C+\epsilon) + p_s^0 - p_i^0 \leq (N-1)(C+\epsilon) + \max_{j \in \mathcal{N}} p_j^0 - \min_{j \in \mathcal{N}} p_j^0.$$

Q.E.D.

The conclusion is that when the problem is feasible, the generic algorithm will terminate with a feasible x and a pair (x,p) satisfying ϵ-CS, as per Prop. 2, and when the problem is infeasible, the generic algorithm will detect infeasibility via one of the three tests (a)-(c) above.

An alternative way to deal with infeasibility is to introduce some artificial arcs to guarantee that the problem is feasible. Each artificial arc should have zero lower flow bound and high cost coefficient. The cost coefficient of each artificial arc should be high enough so that, for a feasible problem, its flow starts and stays at zero in the course of the algorithm. By using the bound of the preceding proposition, we can select the cost coefficients to be high enough so that in the case where the original problem is feasible, the artificial arcs never become ϵ^+-balanced, and their flow stays at zero.

3. SPECIAL CASES OF THE GENERIC ALGORITHM

The price rise operations of the generic algorithm may involve several nodes. When we require that only one node is involved in each price rise, we obtain the ϵ-relaxation method first proposed in [Ber86a], [Ber86b]. We will provide a statement of this method, which is equivalent with the ones given in other sources [Ber86a], [Ber86b], [BeE87], [BeE88], [Gol87], [GoT90]. We will subsequently describe a new algorithm, called *network auction*, which is similar to ϵ-relaxation but occasionally uses price rises involving multiple nodes. In both methods we assume that problem (LNF) is feasible. In practice, the methods should be supplemented with additional mechanisms to detect infeasibility, as discussed at the end of the preceding section.

We use a fixed positive value of ϵ and we start with a pair (x,p) satisfying ϵ-CS. Furthermore, the starting arc flows are integer and it will be seen that the integrality of the arc flows is preserved thanks to the integrality of the node supplies and the arc flow bounds. At the start of a typical iteration of either algorithm we have a flow-price vector pair (x,p) satisfying ϵ-CS and we select a node i with $g_i > 0$; if no such node can be found, the algorithm terminates. During the iteration we perform several operations of the type described in the previous section involving node i and, in the case of the network auction method, neighbor nodes of i (i.e., nodes connected to i with an arc). As these operations are performed, with the attendant price and flow changes, some node surpluses, and push lists are also modified. In our mathematical description of the iteration we assume that these objects are automatically updated at the time when prices and arc flows change. We will discuss appropriate data structures and implementation details later, when we provide a complexity analysis.

The ϵ-Relaxation Method

The typical iteration of the ϵ-relaxation method is as follows:

Typical Iteration of the ϵ-Relaxation Method

Step 0: Select a node i with $g_i > 0$. If no such node exists, terminate the algorithm; else go to Step 1.

Step 1: If the push list of node i is empty go to Step 3; else select an arc a from the push list of i and go to Step 2.

Step 2: Let j be the end-node of arc a, which is opposite to i. Let

$$\delta = \begin{cases} \min\{g_i, c_{ij} - x_{ij}\} & \text{if } a = (i,j), \\ \min\{g_i, x_{ji} - b_{ji}\} & \text{if } a = (j,i). \end{cases} \tag{20}$$

Perform a δ-push of i on arc a. If as a result of this operation we obtain $g_i = 0$, go to Step 3; else go to Step 1.

Step 3: Perform a price rise of node i. If $g_i = 0$ stop; else go to Step 1.

We claim that the above iteration consists of a finite (but positive) number of δ-pushes and a finite (possibly zero) number of price rises, which satisfy conditions (1) and (2) of the generic algorithm. Indeed, since the starting arc flows, the node supplies, and the arc flow bounds are integer, the flow increments δ of all δ-pushes will be positive integers throughout the algorithm. Furthermore, from Eq. (20) it is seen that $\delta \leq g_i$, so condition (1) of the generic algorithm is satisfied. We also note that at most one δ-push per incident arc of node i is performed at each iteration because from Eq. (20) it is seen that a δ-push on arc a in Step 2 is either saturating, which causes arc a to drop out of the push list of i through the end of the iteration, or else results in $g_i = 0$, which leads the iteration to branch to Step 3 and subsequently stop. Therefore, the number of δ-pushes per iteration is finite. In addition we have $g_i > 0$ at the start and $g_i = 0$ at the end of an iteration, so at least one δ-push must occur before an iteration can stop.

Regarding price rises, it is seen that Step 3 can be reached under two conditions:

(a) The push list of i is empty and $g_i > 0$, in which case the price rise in Step 3 will be substantive [in view of the assumption that problem (LNF) is feasible and Lemma 1(b)], and the iteration will branch to Step 1 with the push list of i having at least one new arc.

(b) $g_i = 0$, in which case the iteration will stop following a (possibly trivial) price rise in Step 3.

Thus all price rises involve a node with nonnegative surplus and satisfy condition (2) of the generic algorithm. Since after each substantive price rise with $g_i > 0$, at least one δ-push must be performed, it follows that the number of substantive price rises per iteration is finite. Finally, as argued earlier, there is at most one trivial price rise per iteration. From the preceding observations it is seen that, assuming problem (LNF) is feasible, the ϵ-relaxation method is a special case of the generic algorithm. Furthermore, since the increment δ of each δ-push is integer, it terminates finitely with a feasible flow vector, which is optimal if $\epsilon < 1/N$, cf. Prop. 2.

The Network Auction Algorithm

The network auction algorithm is similar to the ϵ-relaxation method in that it starts from a node i with positive surplus and tries to exhaust the push list of i in preparation for a price rise. However, as it does so, it collects information from neighboring nodes that can be used to effect a price rise involving i and some of its neighbor nodes. The potential advantage here is that the corresponding

price increment may be larger than in the ϵ-relaxation method, thereby saving some iterations, and furthermore a price rise can be performed before the push list of i is exhausted. The potential drawback of the network auction algorithm is that it requires additional overhead relative to the ϵ-relaxation method. Generally, the network auction algorithm seems preferable to the ϵ-relaxation method only for problems with special structure some of which will be discussed in the sequel.

To describe the typical iteration of the network auction algorithm, we need some definitions and a new operation. The *reject capacity* r_i of node i is defined as

$$r_i = \begin{cases} 0, & \text{if the push list } P_i \text{ is empty,} \\ \sum_{\{j|(i,j)\in P_i^+\}}(c_{ij} - x_{ij}) + \sum_{\{j|(j,i)\in P_i^-\}}(x_{ji} - b_{ji}), & \text{otherwise.} \end{cases} \tag{21}$$

Thus, r_i is the sum of the residual capacities of the arcs of the push list P_i.

Definition 4: A *reject operation at node i* consists of performing a saturating push on each of the arcs in the push list of i.

Note that in a reject operation at node i, the push list of i is emptied and the total amount of flow "pushed away" from i is equal to the reject capacity r_i.

The network auction iteration uses a subset L of neighbor nodes of i, which is empty at the start of the iteration. The nodes in L are the ones whose push list is emptied during the iteration through a reject operation. As a result, the prices of all nodes in L can be increased at the end of the iteration. This will occur regardless of whether the price of i is also increased. The price increase of the nodes in L, however, often has the beneficial effect of allowing a larger price increase for i than would otherwise be possible. On the other hand, a network auction iteration for which the set L stays empty, is identical with a corresponding ϵ-relaxation iteration.

Typical Iteration of the Network Auction Algorithm

Step 0: Select a node i with $g_i > 0$. If no such node exists, terminate the algorithm; else set $L = \emptyset$ and go to Step 1.

Step 1: Let P' be the set of arcs of the push list of i whose end-node opposite to i does not belong to L. If P' is empty go to Step 3; else select an arc a from P' and go to Step 2.

Step 2: Let j be the end-node of arc a, which is opposite to i. If $r_j \leq g_j$, perform a reject operation at node j, set $L := L \cup \{j\}$, and go to Step 1. Else let

$$\delta = \begin{cases} \min\{r_j - g_j, g_i, c_{ij} - x_{ij}\} & \text{if } a = (i,j), \\ \min\{r_j - g_j, g_i, x_{ji} - b_{ji}\} & \text{if } a = (j,i). \end{cases} \tag{22}$$

If $\delta = r_j - g_j$, perform a δ-push of i on arc a, perform a reject operation at node j, and set $L := L \cup \{j\}$; else just perform a δ-push of i on arc a. If as a result of these operations we obtain $g_i = 0$ go to Step 3; else go to Step 1.

Step 3: Perform a price rise of the set $\{i\} \cup L$. Then, if $L \neq \emptyset$, perform a price rise of L. Then, if $g_i = 0$ stop; else set $L = \emptyset$ and go to Step 1.

An alternative form of Step 3 is the following:

Alternative Form of Step 3: Perform a price rise of the set $\{i\} \cup L$. Then, if $L \neq \emptyset$, sequentially perform a price rise of each of the nodes in L. Then, if $g_i = 0$ stop; else set $L = \emptyset$ and go to Step 1.

It can be shown that the above alternative form of Step 3 leads to larger price rises for transportation problems than the first form, because for bipartite graphs, there is no arc joining any pair of nodes in L. Therefore, the alternative form of Step 3 is preferable for bipartite problems, or more generally, in cases where for most iterations there is no arc connecting any two nodes of L.

It is also possible to consider variations of the network auction algorithm whereby an iteration consists of a finite sequence of network auction and ϵ-relaxation iterations, in any order. Such combined iterations may be tailored to particular graph structures, thereby resulting in simpler implementations.

Using similar reasoning as for the ϵ-relaxation iteration, we can show that the network auction algorithm is a special case of the generic algorithm. Indeed each iteration consists of δ-pushes, reject operations, and price rises, and the δ increments of all δ-pushes are positive integers. From Eq. (22) it is seen that $\delta \leq g_i$, while we have $r_j \leq g_j$ whenever a node j enters the set L and a reject operation is performed at j; this means that following a δ-push or a reject operation, the surplus of the corresponding node is nonnegative, so condition (1) of the generic algorithm is satisfied. Note also that the argument of the proof of Prop. 2 can be adapted to show that the number of δ-pushes per iteration is finite. Furthermore, since we have $g_i > 0$ at the start and $g_i = 0$ at the end of an iteration, it follows that at least one δ-push must occur before the iteration can stop.

Regarding price rises, we note that they involve nodes with nonnegative surplus, thereby satisfying condition (2) of the generic algorithm. To show that the number of substantive price rises per iteration is finite, note that with each substantive price rise, the reject capacity of either node i or a neighbor node of i (belonging to L) is increased by an integer amount. It follows that the number of substantive price rises per iteration cannot be infinite, since the reject capacity of each node is bounded and the number of δ-pushes per iteration is finite. Finally, regarding the number of trivial price rises per iteration, we note that the first price rise in Step 3 involving the set $\{i\} \cup L$ will be trivial only if the modified push list P' (cf. Step 1) is nonempty (the push lists of all nodes in L are empty following the reject operation in Step 2), in which case we must have $g_i = 0$ and the iteration will stop at that visit to Step 3. Therefore with each visit to Step 3 except at most one, there will be at least one substantive price rise. Since the number of substantive price rises is finite, it follows that the number of visits to Step 3 is finite, implying that the number of trivial price rises is also finite. Thus, the network auction algorithm is a special case of the generic algorithm and performs at least one δ-push per iteration. Therefore, Prop. 2 guarantees the termination of the algorithm with an optimal flow vector obtained if $\epsilon < 1/N$.

Note that if the first price rise involving the set $\{i\} \cup L$ in Step 3 is trivial and L is nonempty, the subsequent price rise in Step 3 (or price rises, if the alternative form of Step 3 is used) involving the set L will be substantive, since following the reject operation in Step 2, the push lists of all the nodes in L are empty. Thus, with each visit to Step 3 for which the set L nonempty, there is a price increase of all the nodes of L. Practical experience, as well as the complexity analysis of the next section, suggest that high frequency and large size of price rises is a good performance indicator, so the extra work needed to compute the set L may be compensated by the associated extra price rises.

Relation to the Auction Algorithm for the Assignment Problem

We now show how the assignment auction algorithm of [Ber79] is a special case of the network auction algorithm. (It is also possible to show that the assignment auction algorithm can be obtained as a special case of the ϵ-relaxation method; see [BeE88], [BeT89], [Ber91a].) Consider the assignment problem where the graph is bipartite, having n nodes i with $s_i = 1$, called *persons*, and n nodes j with $s_j = -1$, called *objects*. All arcs (i, j) connect persons i with objects j, and the arc flow bounds are $b_{ij} = 0$ and $c_{ij} = 1$. We choose initially x and p with the following properties:

(a) x and p satisfy ϵ-CS.

(b) For all arcs (i, j), either $x_{ij} = 0$ or $x_{ij} = 1$. Furthermore if $x_{ij} = 1$, then the arc (i, j) belongs to the push list of j.

(c) $g_i = 0$ or $g_i = 1$ for all persons i and $g_j = 0$ or $g_j = -1$ for all objects j.

It will be seen that the algorithm preserves these properties. The typical iteration of the network auction algorithm starts with a person i with $g_i = 1$ and if the push list of i is empty, it raises p_i to

$$p_i = \min_{\{j|(i,j)\in A\}} \{a_{ij} + p_j + \epsilon\} \tag{23}$$

(Step 3); then chooses an arc (i, j) from the push list of i, increases x_{ij} to 1 (this is a δ-push with $\delta = 1$, which assigns person i to object j); then if j was assigned prior to the iteration to some person i', the iteration reduces $x_{i'j}$ to 0 [this is the reject operation at node j, canceling the assignment of j to i', which is possible because (i', j) belongs to the push list of j by property (b) above]. The price rises of Step 3 are now performed as follows with L being the singleton set $\{j\}$: the prices of i and j are raised by an increment

$$\min_{\{k|(i,k)\in A, k\neq j\}} \{a_{ik} + p_k + \epsilon\} - p_i \tag{24}$$

(this is the price rise of the set $\{i\} \cup L$), and then the price of j is raised by 2ϵ (this is the price rise of L). Since at this point we have $g_i = 0$, the iteration stops. It can be seen that at the end of the iteration, ϵ-CS holds, the arc (i,j) whose flow was changed from $x_{ij} = 0$ to $x_{ij} = 1$, is ϵ-unblocked, and the property $g_i = 0$ or $g_i = 1$ for all persons i and $g_j = 0$ or $g_j = -1$ for all objects j is preserved. This allows the preceding iteration to be repeated as long as there are unassigned persons and objects. If we view $a_{ik} + p_k$ as the cost (negative value) of object k for person i, it can be seen that object j was chosen as the object of minimum cost for person i at the start of the iteration. From Eqs. (23) and (24), it is seen that the price of j was raised so that its cost is equal to 2ϵ plus the minimum object cost for i at the end of the iteration. This leads to the auction interpretation, whereby a person selects its best object and bids up its price as much as possible, subject to the constraint that the cost of the object is within 2ϵ from the minimum cost of other objects should the bid be accepted. We refer to [Ber88], [BeE88], [BeC89a], [Ber91a], and [BCT91] for further discussion and recent extensions of the assignment auction algorithm.

Relation to the Transportation Auction Algorithm

The transportation auction algorithm of [BeC89a] can be shown to be a variation of the network auction algorithm, which exploits the special structure of transportation problems. For these problems, the graph is bipartite, having n nodes i with positive supply s_i, called sources, and n nodes j with negative supply $-d_j$, called sinks. All arcs (i,j) connect sources i with sinks j and the arc flow bounds are $b_{ij} = 0$ and $c_{ij} = \min\{s_i, d_j\}$. We now describe the steps in the transportation auction algorithm of [BeC89a] and simultaneously indicate how they can be related to steps of a combined version of the network auction and ϵ-relaxation methods.

The initial x and p are required to have the following properties:

(a) x and p satisfy ϵ-CS.

(b) For all arcs (i,j), $x_{ij} \in [0, c_{ij}]$.

(c) $0 \leq g_i \leq s_i$ for all sources i, and $-d_j \leq g_j \leq 0$ for all sinks j.

(d) For any sink j, we have either $r_j = 0$ or $g_j = 0$.

These properties are preserved throughout the algorithm.

An iteration is started with a source i with $0 < g_i$ [by properties (b) and (c) above, if no such source can be found, x is feasible and the algorithm terminates]. If the push list of i is empty, we first raise p_i to

$$p_i = \min_{\{j \mid (i,j) \in \mathcal{A}\}} \{a_{ij} + p_j + \epsilon\},$$

(this is a Step 3 with $L = \emptyset$ in the network auction algorithm). Then, we choose an arc (i, j) from the push list of i and compute

$$\delta = \min\{r_j - g_j, g_i, c_{ij} - x_{ij}\},$$

[cf. Eq. (22)]. In the language of the transportation auction algorithm of [BeC89a], source i "bids" for δ units of the demand of sink j [this is equivalent to a δ-push at i on arc (i, j)]. If the bid size δ is large enough to exhaust the reject capacity of the sink (i.e., if $\delta = r_j - g_j$), a reject operation is performed at sink j (Step 2), and sink j is added to a list L of sinks. If the resulting surplus g_i is still positive, the above bidding operation is repeated until $g_i = 0$ or until the push list of i is empty. At the end of this step, the set L (if nonempty) will consist of some sinks j with $x_{ij} > 0$.

The next step in the transportation auction algorithm of [BeC89a] is to raise the prices and construct new push lists for the source i and the sinks in L (cf. the alternative form of Step 3 of the network auction algorithm). This is done in a manner which is equivalent to a price rise on the set $i \cup L$, followed by a price rise on each sink in L (if $L \neq \emptyset$). Through this step, the price of each sink $j \in L$ is raised to the highest level for which its push list is nonempty.

In the above construction, it is possible that there are sinks j for which x_{ij} was increased during the bidding iteration of source i (cf. Step 2 of the network auction algorithm), but which were never included in the set L. In order to preserve the property $g_j \leq 0$ for all sinks j, the transportation auction algorithm performs δ-pushes as in Step 2 of the ϵ-relaxation method, repeatedly until the surplus of each sink $j \notin L$ with $g_j > 0$ is shifted to sources in the push list of j, and g_j is set to zero; such sources must exist, for otherwise j would have been included in the set L. In this way, all bidding iterations will start with source nodes, simplifying the implementation.

Following the above flow changes and price rises, we check whether $g_i > 0$, in which case L is reset to empty, and the iteration is in effect restarted with node i (i.e. we return to Step 1 of the network auction algorithm); otherwise the iteration stops.

After the iteration has stopped, conditions (a) and (b) above are satisfied because all of the steps are part of either the ϵ-relaxation iteration or the network auction iteration. Furthermore, conditions (c) and (d) are maintained by the sequence of operations described.

4. AN ALGORITHM FOR THE K NODE-DISJOINT SHORTEST PATH PROBLEM

In this section we consider a generalization of the single origin/single destination shortest path problem, where instead of a single path, we seek k node-disjoint paths that minimize a linear cost.

An example is a three-dimensional assignment problem, involving the optimal choice of k disjoint ordered triplets, where the cost of a triplet (i, j, m) is separable of the form $a_{ij} + a_{jm}$. We derive a specialized version of the network auction algorithm for this problem. Note that in the literature, the term "k shortest path problem" has been used somewhat differently; it refers to finding the shortest, second shortest, etc., up to kth shortest path between an origin and a destination [Dre69].

Suppose that we are given a graph with node set $\mathcal{N}$, arc set $\mathcal{A}$, and integer arc costs a_{ij}. In this section, by a *path* P we mean either a single node i (in which case we say that P is a *trivial* path), or else a sequence of arcs $(i_1, i_2), (i_2, i_3), \ldots, (i_{m-1}, i_m)$. If the nodes $i_1, \ldots, i_m$ are distinct we say that the path is *simple*. We refer to i_1 as the *starting* node of P and to i_m as the *terminal* node of P; if P is trivial, its unique node is viewed as both the starting and the terminal node of P. The *cost of a nontrivial path* P is the sum of the costs of its arcs. By a *cycle* we mean a sequence of arcs $(i_1, i_2), (i_2, i_3), \ldots, (i_{m-1}, i_1)$. If the nodes $i_1, \ldots, i_{m-1}$ are distinct we say that the cycle is *simple*.

Let s and t be given nodes called the *origin* and the *destination*, respectively. We assume that:

(a) s has no incoming arcs, t has no outgoing arcs, and (s, t) is not an arc. Furthermore, each node except for t has at least one outgoing arc. (These assumptions are convenient for stating the algorithm but do not involve a loss of generality.)

(b) The cost of each cycle is positive.

For a given positive integer k, we want to find k nontrivial simple paths $P_1, P_2, \ldots, P_k$ that start at s, terminate at t, and satisfy the following conditions:

(a) The paths are node-disjoint, that is, any pair of paths from the set $\{P_1, P_2, \ldots, P_k\}$ shares no node other than s and t.

(b) The sum of the costs of $P_1, P_2, \ldots, P_k$ is minimal.

It is possible to view this problem as a special case of the minimum cost flow problem (LNF) by replacing each node i other than s and t with two nodes i^+ and i^-, which are connected with a zero cost arc (i^+, i^-), and by replacing each arc (i, j) with the arc (i^-, j^+) of cost a_{ij}, as shown in Fig. 2. All arcs have feasible flow range $[0, 1]$. The supply of the origin is k, the supply of the destination is $-k$, and the supply of every other node is zero.

The following algorithm can be obtained by applying in a particular way the network auction algorithm to the above minimum cost flow problem. In particular, there will be price rises of pairs or triplets of nodes [either i^+ and i^-, or i^- and j^+ where (i, j) is an arc, or i^+, i^-, and j^+ where (i, j) is an arc]. These two-node or three-node price rises are almost as easy as single node price rises, and the algorithm is far more efficient than what would be obtained by straighforward use of

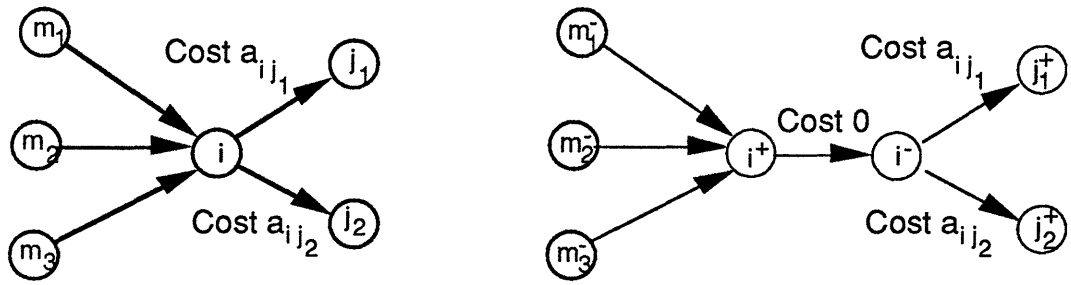

Figure 2 Converting the k node-disjoint shortest path problem to a minimum cost flow problem with all arcs having feasible flow range $[0,1]$. Each node i is split into the two nodes i^+ and i^-, which are connected with a unit capacity and zero cost arc. Each arc (i,j) is replaced by an arc (i^-,j^+) of cost a_{ij}.

the ϵ-relaxation method.

To simplify the presentation, we will describe the algorithm from first principles, and we will indicate more precisely the connections with the network auction algorithm later. We first introduce a price and flow vector structure, and a corresponding definition of ϵ-CS, which are adapted to the k node-disjoint shortest path problem. This form of ϵ-CS is somewhat more restrictive than the form given in Section 2.

ϵ-CS for the k Node-Disjoint Shortest Path Problem

The subsequent k node-disjoint shortest path algorithm maintains the following:

(a) Two prices p_i^+ and p_i^- for each node $i \neq s,t$, which satisfy

$$p_i^- \leq p_i^+, \qquad \forall\, i \neq s,t; \tag{25}$$

these prices correspond to the constituent nodes i^+ and i^- referred to earlier.

(b) A price p_s^- for the origin and a price p_t^+ for the destination, which are specified in terms of the remaining prices by the equations

$$p_s^- = \min\{z \mid z \geq a_{sj} + p_j^+ + \epsilon \text{ for } k \text{ or more arcs } (s,j)\}, \tag{26}$$

$$p_t^+ = \max\{z \mid a_{it} + z \leq p_i^- + \epsilon \text{ for } k \text{ or more arcs } (i,t)\}. \tag{27}$$

(c) A set of simple paths $P_1,\ldots,P_m$ and a set of simple cycles $C_1,\ldots,C_n$, which are all node-disjoint, and a flow vector x such that for all arcs (i,j)

$$x_{ij} = \begin{cases} 1 & \text{if } (i,j) \text{ belongs to one of the paths } P_1,\ldots,P_m \text{ or cycles } C_1,\ldots,C_n, \\ 0 & \text{otherwise.} \end{cases} \tag{28}$$

We require that out of the paths $P_1, \ldots, P_m$, exactly k are nontrivial and start at the origin, and at most k are nontrivial and terminate at the destination. Furthermore, a trivial path consisting of a single node, say i, belongs to the set $\{P_1, \ldots, P_m\}$ if and only if $i \neq s, t$, $p_i^- < p_i^+$, and no nontrivial path from $\{P_1, \ldots, P_m\}$ or cycle from $\{C_1, \ldots, C_n\}$ passes through i. [Note that the triplet (x, p^+, p^-) specifies completely the paths $P_1, \ldots, P_m$ and the cycles $C_1, \ldots, C_n$ based on the above requirements.]

We say that the triplet (x, p^+, p^-) *satisfies ϵ-CS* for the k node-disjoint shortest path problem if the above conditions hold and in addition

$$p_i^- \geq a_{ij} + p_j^+ - \epsilon, \qquad \forall\ (i,j) \text{ such that } x_{ij} = 1, \tag{29a}$$

$$p_i^- \leq a_{ij} + p_j^+ + \epsilon, \qquad \forall\ (i,j) \text{ such that } x_{ij} = 0. \tag{29b}$$

For a triplet (x, p^+, p^-) satisfying ϵ-CS, we say that one of the corresponding paths $P_1, \ldots, P_m$ is *active* if it terminates at a node other than the destination. Note that a trivial path consisting of a single node $i \neq s, t$ is active if and only if $p_i^- < p_i^+$. Note also that *if there are no active paths*, then in view of the requirement that out of the paths $P_1, \ldots, P_m$, exactly k are nontrivial and start at the origin, and no more than k are nontrivial and terminate at the destination, *the paths $P_1, \ldots, P_m$ must be k in number, must all start at s, and must all terminate at t, thereby yielding a feasible solution of the k node-disjoint shortest path problem.*

The following proposition gives the basis for the subsequent algorithm.

Proposition 4: Suppose that the triplet (x, p^+, p^-) satisfies ϵ-CS. Then if $\epsilon < 1/N$, there are no simple cycles corresponding to (x, p^+, p^-). If in addition none of the corresponding paths $P_1, \ldots, P_m$ is active, then these paths constitute an optimal solution of the k node-disjoint shortest path problem.

Proof: If C is a simple cycle corresponding to (x, p^+, p^-), then for every arc (i,j) of C we must have $x_{ij} = 1$, and from Eqs. (25) and (29),

$$p_i^+ \geq p_i^- \geq a_{ij} + p_j^+ - \epsilon.$$

By adding this relation over all arcs of C, we obtain

$$\text{Cost of } C = \sum_{(i,j) \in C} a_{ij} \leq (N-1)\epsilon.$$

Since the arc costs are integer and $\epsilon < 1/N$, it follows that the cost of C is less or equal to zero, which contradicts our assumption that all cycle costs are positive.

If in addition there are no active paths, the vector x is a feasible solution that together with the price vector (p^+, p^-) satisfies ϵ-CS for the associated minimum cost flow problem, cf. Fig. 2. The

optimality proof for x is obtained by adapting the proof of Prop. 1 (see e.g. [BeT89] or [Ber91a]) and by using the fact $p_i^+ \geq p_i^-$ for all $i \neq s,t$. We omit the details. **Q.E.D.**

The k node-disjoint shortest path algorithm starts each iteration with a triplet (x, p^+, p^-) satisfying ϵ-CS. The algorithm terminates if there is no active path. Otherwise, the algorithm selects an active path, and either *contracts* it by deleting its terminal node, or *extends* it by connecting its terminal node to another node; also the triplet (x, p^+, p^-) and at most one other of the corresponding paths and cycles are modified while maintaining ϵ-CS. As a result of the iteration, the path may get eliminated (if it consists of a single node or arc and is contracted) or may stop being active (if it joins a path that terminates at the destination). The number of active paths then decreases by one. It is also possible that the number of active paths stays the same as a result of the iteration.

To start the algorithm, we need an initial triplet (x, p^+, p^-) satisfying ϵ-CS. One way to obtain such a triplet is as follows:

Standard Initialization

Set $x_{ij} = 0$ for all arcs (i,j), select p_i^+ arbitrarily for all $i \neq s$, set

$$p_i^- = \min \left\{ p_i^+, \min_{\{j \mid (i,j) \in \mathcal{A}\}} \{a_{ij} + p_j^+\} + \epsilon \right\}, \qquad \forall\, i \neq s; \tag{30}$$

and set p_s^- and p_t^+ according to Eqs. (26) and (27); then select k nodes j such that $(s,j) \in \mathcal{A}$ and $p_s^- \geq a_{sj} + p_j^+ + \epsilon$, and for all these nodes, set $x_{sj} = 1$ and $p_j^+ = p_s^- - a_{sj} + \epsilon$; then set $x_{it} = 1$ for all nodes arcs (i,t) with $p_i^- > a_{it} + p_t^+ + \epsilon$.

Contraction and Extension Operations

We now describe the operations of contraction and extension of an active path. Let (x, p^+, p^-) be a triplet satisfying ϵ-CS, and let $P_1, \ldots, P_m$ and $C_1, \ldots, C_n$ be the corresponding simple paths and cycles. Suppose that P is an active path with terminal node i.

A *contraction* operation for P can be performed in one of the following two circumstances:

(a) P consists of just node i and

$$p_i^+ < \min_{\{j \mid (i,j) \in \mathcal{A}\}} \{a_{ij} + p_j^+\} + \epsilon, \tag{31}$$

in which case the contraction consists of setting

$$p_i^+ = p_i^- = \min_{\{j \mid (i,j) \in \mathcal{A}\}} \{a_{ij} + p_j^+\} + \epsilon. \tag{32}$$

(In this case P is eliminated as a path.)

(b) P has a final arc, say (r,i), and

$$p_r^- - a_{ri} < \min_{\{j|(i,j)\in\mathcal{A}\}}\{a_{ij}+p_j^+\}, \tag{33}$$

in which case the contraction consists of setting

$$p_i^+ = p_i^- = \min_{\{j|(i,j)\in\mathcal{A}\}}\{a_{ij}+p_j^+\}+\epsilon,$$

deleting the final arc (r,i) from P, and setting $x_{ri}=0$. If r is the origin node s, the following additional operations are executed: the price p_s^- is set to the value given by Eq. (26) [this value may be higher than the previous value of p_s^- since p_i^+ was just increased]; also an arc (s,n) is found such that $x_{sn}=0$ and $p_s^- = a_{sn}+p_n^++\epsilon$, and its flow x_{sn} is set to 1, while the flow of each incoming arc (r,n) with $r\neq s$ is set to zero. [This creates a new nontrivial path starting at the origin, to replace the path P consisting of the arc (s,i) that was eliminated through the contraction.] Following these changes, the price p_n^+ of each node n with $x_{sn}=1$ is set to $p_s^- - a_{sn}+\epsilon$.

An *extension* operation for P is performed only if a contraction is not possible. Then we find a node j_i such that

$$j_i = \arg\min_{\{j|(i,j)\in\mathcal{A}\}}\{a_{ij}+p_j^+\},$$

and we also find

$$w_i = \begin{cases} \min_{\{j|j\neq j_i,\ (i,j)\in\mathcal{A}\}}\{a_{ij}+p_j^+\}+\epsilon & \text{if } i \text{ has two or more outgoing arcs,}\\ \infty & \text{if } (i,j_i) \text{ is the only outgoing arc from } i,\end{cases}$$

$$v_i = \begin{cases} p_r^- - a_{ri}+\epsilon & \text{if } P \text{ has a final arc } (r,i),\\ p_i^+ & \text{if } P \text{ consists of just node } i.\end{cases}$$

We then distinguish three cases, depending on whether j_i is the destination node, and whether an arc connecting the origin with j_i is part of a current path ($x_{sj_i}=1$).

(a) If $j_i\neq t$ and $x_{sj_i}=0$, the prices p_i^+ and p_i^- are set to

$$p_i^+ = v_i, \qquad p_i^- = \min\{v_i,w_i\}.$$

Furthermore, the price $p_{j_i}^+$ is set to

$$p_{j_i}^+ = \min\{v_i,w_i\} - a_{ij_i}+\epsilon, \tag{34}$$

while the arc (i,j_i) is added to P and its flow is set to 1; also the flow of any incoming arc (n,j_i) with $n\neq i$ and $x_{nj_i}=1$ is set to 0 (this could make n the terminal node of an active path).

(b) If $j_i \neq t$ and $x_{sj_i} = 1$, all the operations of the preceding case (a) are performed, including setting x_{sj_i} to 0. The following additional operations are then executed to create a new nontrivial path starting at the origin, replacing the path $P = (s, j_i)$ that was just eliminated [cf. case (b) of the contraction operation]: the price p_s^- is set to the value given by Eq. (26); also an arc (s, n) is found such that $x_{sn} = 0$ and $p_s^- = a_{sn} + p_n^+ + \epsilon$, and its flow x_{sn} is set to 1, while the flow of each incoming arc (r, n) with $r \neq s$ is set to zero. Following these changes, the price p_n^+ of each node n with $x_{sn} = 1$ is set to $p_s^- - a_{sn} + \epsilon$.

(c) If $j_i = t$, the prices p_t^+ and p_t^- are set to

$$p_t^+ = v_i, \qquad p_t^- = \min\{v_i, w_i\},$$

and the arc (i, t) is added to P, while its flow is set to 1. If as a result, the number of paths terminating at t is $k + 1$, the price p_t^+ is set to the value given by Eq. (27), and an arc (n, t) is found such that $x_{nt} = 1$ and $p_t^+ = p_n^- - a_{nt} + \epsilon$, and its flow x_{nt} is set to 0. [This eliminates a nontrivial path terminating at the destination, and since P was extended by arc (i, t), the number of nontrivial paths terminating at the destination is maintained at k.]

Note that in an extension operation it is possible that the extension node j_i is already part of P; then by setting $x_{ij_i} = 1$, a cycle C is obtained that consists of the portion of P between j_i and i and the arc (i, j_i). In this case, if j_i is the starting node of P, the active path P is replaced by the cycle C, and the number of active paths is reduced by one. Otherwise, the portion of P up to but not including j_i may become an active path.

By examining the nature of the contraction and extension operations, it is straightforward to verify the following:

(a) At the start of each iteration, the triplet (x, p^+, p^-) satisfies ϵ-CS.

(b) A contraction or extension that does not change the flow of any of the outgoing arcs from the origin is equivalent to an iteration of the network auction algorithm applied to the associated minimum cost flow problem described earlier.

(c) A contraction or extension that changes the flow of an outgoing arc from the origin [cf. case (b) of a contraction or case (b) of an extension] is equivalent to two iterations of the network auction algorithm: an iteration starting at node i followed by an iteration starting at the origin.

k Node-Disjoint Shortest Path Algorithm

Our algorithm starts each iteration with a triplet (x, p^+, p^-), and corresponding simple paths and cycles $P_1, \ldots, P_m$, $C_1, \ldots, C_n$ satisfying ϵ-CS.

Typical Iteration of the k Node-Disjoint Shortest Path Algorithm

Select an active path P. If no such path exists, terminate the algorithm; else if a contraction is possible for P [that is, if the corresponding condition (31) or (33) holds] perform the contraction, and otherwise perform an extension of P.

Figure 3 illustrates the algorithm for a simple example. From our earlier discussion, it is seen that the algorithm is a special case of the network auction algorithm. By using Prop. 2, it follows that for a feasible problem, the algorithm terminates, and by Prop. 4, the feasible solution obtained at termination is optimal if $\epsilon < 1/N$.

It is interesting to note that a $k \times k$ assignment problem can be converted to a k node-disjoint shortest path problem by adding an origin node s, which is connected with each person node with a zero cost arc, and by also adding a destination node t, which is connected to each object node with a zero cost arc. It can be verified that when the algorithm of this section is specialized to this problem, it becomes equivalent to the auction algorithm for the assignment problem.

For another interesting connection, consider the case $k = 1$. Then the problem becomes a single origin/single destination shortest path problem. It can be verified that when the algorithm of this section is specialized to this problem but with the important difference that $\epsilon = 0$, it becomes equivalent to a recently proposed auction algorithm for shortest paths [Ber91a], [Ber91b].

5. COMPLEXITY ANALYSIS

In this section, we derive a bound on the order of time required by a simple implementation of the network auction algorithm. We then introduce a scaled version of the algorithm with a $O(N^3 \log(NC))$ worst-case time bound. Our analysis parallels a corresponding analysis of the ϵ-relaxation method given in [BeE87], [BeE88], and [BeT89], which in turn uses the sweep implementation ideas of [Ber86a] and some of the scaling ideas of [Gol87].

Our method of analysis of the network auction algorithm yields also an $O(N^4 \log(NC))$ bound for a straightforward implementation of the generic auction algorithm, and an $O(NA \log(NC))$ bound for the case where the feasible flow range of all arcs is $[0, 1]$.

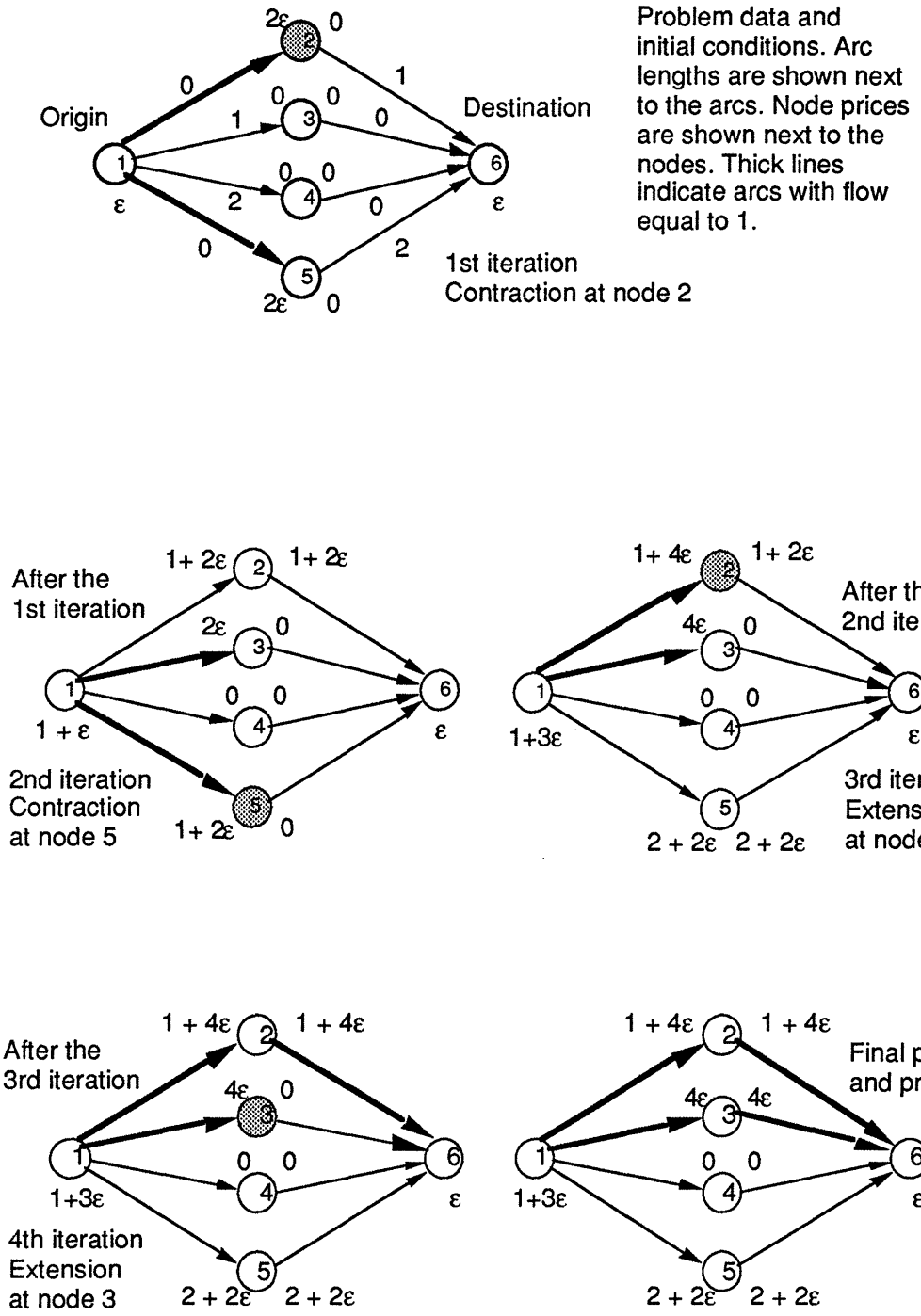

Figure 3 Illustration of the k node-disjoint shortest path algorithm for $k = 2$, starting with $p_i^+ = 0$ for all $i \neq s$ and using the standard initialization. The problem data is given in the first graph. The numbers on the left and the right sides of a node i are the prices p_i^+ and p_i^-, respectively. Thick (thin) line arcs are the ones with flow equal to 1 (0, respectively). Initially the active paths are $(1,2)$ and $(1,5)$. At the start of the second and third iterations there is only one active path, $(1,5)$ and $(1,2)$, respectively.

We first make some assumptions:

Assumption 1: Problem (LNF) is feasible.

Assumption 2: All arc cost coefficients are integer multiples of ϵ.

Assumption 3: All starting prices are integer multiples of ϵ, all starting flows are integer, and together they satisfy ϵ-CS.

We assume that the push lists of the nodes, are maintained in a data structure that makes possible the addition and deletion of a single arc in $O(1)$ time; this is true, for example if each push list P_i is maintained in a doubly linked list. Then it is seen that selecting an arc in Step 1 takes $O(1)$ time, updating the push list of node i following a δ-push in Step 2 takes $O(1)$ time per δ-push, and updating the push list of a node i following a price rise involving node i in Step 3 takes $O(d_i)$ time per price rise and node, where d_i is the number of incident arcs of node i.

The Admissible Graph

A notion that is central in our analysis is the so called *admissible graph*, introduced in [Ber86a], which consists of the push list arcs, except that the directions of those arcs that are incoming to the corresponding nodes are reversed to make them consistent with the direction in which flow is pushed in the network auction algorithm. Formally, the admissible graph is defined as $G^* = (N, A^*)$, where A^* contains arc (i,j) if either (i,j) is an ϵ^+-unblocked arc, or (j,i) is ϵ^--unblocked arc. Note that the admissible graph depends on the current pair (x,p) that satisfies ϵ-CS and changes as the pair (x,p) changes during the course of the algorithm. In particular, when there is a saturating push on an arc, the arc is removed from A^*. However, δ-pushes cannot create any new arcs of the admissible graph. Furthermore, when there is a substantive price rise of a node set I in Step 3, all the arcs (i,j) and (j,i) with $i \in I$ and $j \notin I$ that belonged to A^* prior to the price rise are removed from A^*, and an arc (i,j) is added to A^* if either an arc (i,j) (or (j,i)) with $i \in I$ and $j \notin I$ became ϵ^+-unblocked (or ϵ^--unblocked, respectively), as a result of the price rise. Thus following the price rise, there are no arcs (j,i) of the admissible graph that have a start node $j \notin I$ and an end node $i \in I$, leading to the conclusion that price rises cannot create any new cycles of the admissible graph (this will be shown more precisely in the proof of the subsequent Prop. 5). Our next assumption is that:

Assumption 4: Initially, the admissible graph has no arcs.

Assumption 4 can be satisfied by setting initially $x_{ij} = c_{ij}$ (or $x_{ij} = b_{ij}$) for all arcs (i,j) with $p_i = p_j + a_{ij} + \epsilon$ ($p_i = p_j + a_{ij} - \epsilon$, respectively). Under this assumption, the admissible graph is initially acyclic and since, based on the preceding arguments, neither δ-pushes nor price rises can

create a cycle, we conclude that *the admissible graph is acyclic throughout the algorithm.* (Again this will be shown formally as part of the proof of Prop. 5.)

The Sweep Implementation

In order to obtain the subsequent complexity bounds, we need a certain rule for choosing the starting node in each iteration. This rule is the basis for the sweep implementation referred to earlier, and uses an ordered list T of all the nodes, which is restructured repeatedly in the course of the algorithm. The initial choice of the list can be arbitrary. We say that node i *ranks higher* (or *lower*) than node j at some time, if the position of i in the list T is higher (or lower, respectively) than the one of j at that time. The order of nodes in the list will be shown to be related to the admissible graph (see the proof of the subsequent Prop. 5). In particular, it will be seen that a node i ranks higher than all nodes j such that there is a directed path from i to j in the admissible graph.

The order of nodes in T is changed only when there is a substantive price rise in Step 3. In particular, if the price rise involves a set I, the nodes of I are placed at the top of T in the order in which they appear in T prior to the price rise. The position of the nodes not in I is not changed. Figure 4 illustrates the rule for restructuring the list T following a price rise. We note that the restructuring of T can be done in $O(N)$ time per substantive price rise. In the case where the alternative form of Step 3 of the network auction algorithm or Step 3 of the ϵ-relaxation algorithm is used, the restructuring of T can be done more simply, in time $O(1)$ per single node price increase, by placing sequentially the nodes of L at the top of T as their price is increased. In practice one may want to maintain T in an appropriate data structure, such as a linked list, to minimize the restructuring overhead, but this is not necessary for the subsequent complexity bounds.

If a given iteration is started at node i, the list T is used to select the starting node i' for the next iteration as follows:

Let N_i be the set of nodes that were ranking lower than i in T at the start of the given iteration and whose price did not change during the iteration. If N_i contains nodes that have positive surplus at the end of the iteration, then i' is chosen to be the highest ranking of these nodes; otherwise i' is chosen to be the highest ranking node in T among all the nodes that have positive surplus at the end of the iteration. (Thus, the algorithm goes down the list T selecting nodes with positive surplus and when it reaches the bottom of the list, it returns to the top of the list.)

A sequence of iterations between two successive times that the algorithm starts an iteration with the highest ranking node with positive surplus is called a *cycle*. Note that the computation time for selecting the starting node of an iteration by going down the list T and checking for a positive surplus node, is $O(N)$ per cycle. Our final assumption is:

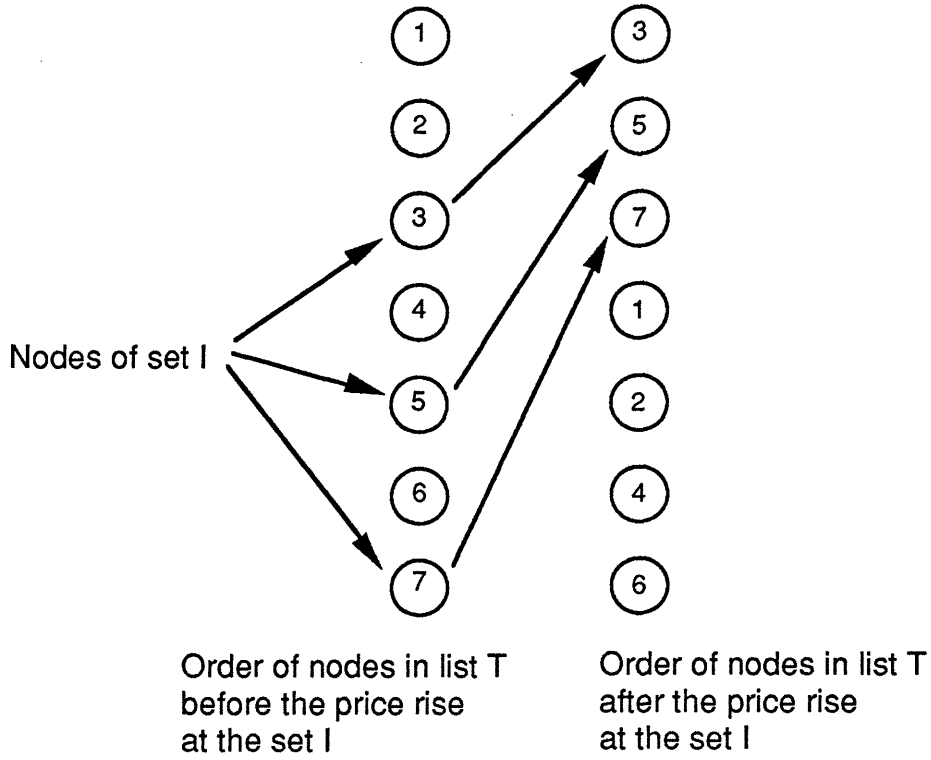

Figure 4 Illustration of the rule for restructuring the order of of nodes in the list T following a price rise at the set I.

Assumption 5: The starting nodes of iterations of the network auction algorithm are chosen as described above.

Main Result

We begin the complexity analysis by introducing some terminology. For any path H, we denote by $s(H)$ and $t(H)$ the start and end nodes of H, respectively, and by H^+ and H^- the sets of forward and backward arcs of H, respectively, as the path is traversed in the direction from $s(H)$ to $t(H)$. We say that the path is *simple* if it has no repeated nodes. For any price vector p and simple path H, we define

$$d_H(p) = \max\left\{0, \sum_{(i,j)\in H^+}(p_i - p_j - a_{ij}) - \sum_{(i,j)\in H^-}(p_i - p_j - a_{ij})\right\}$$

$$= \max\left\{0, p_{s(H)} - p_{t(H)} - \sum_{(i,j)\in H^+} a_{ij} + \sum_{(i,j)\in H^-} a_{ij}\right\}. \qquad (35)$$

Note that the second term in the maximum above may be viewed as a "reduced cost length of H," being the sum of the reduced costs $(p_i - p_j - a_{ij})$ over all forward arcs of H minus the sum of $(p_i - p_j - a_{ij})$ over all backward arcs of H. It is seen that the following upper bound on $d_H(p)$ holds:

$$d_H(p) \leq p^+ - p^- + L, \tag{36}$$

where $p^+ = \max_i p_i$, $p^- = \min_i p_i$, and L is the maximum simple path length, where the length of each arc (i,j) is taken to be $|a_{ij}|$. Since any simple path can have at most $N-1$ arcs, it is seen that when $p^+ - p^- = O(1)$, we have $d_H(p) = O(NC)$.

For any capacity-feasible flow vector x, we say that a simple path H is *unblocked with respect to* x if we have $x_{ij} < c_{ij}$ for all arcs $(i,j) \in H^+$ and we have $x_{ij} > b_{ij}$ for all arcs $(i,j) \in H^-$.

For any price vector p and feasible flow vector x, denote

$$D(p,f) = \max\{d_H(p) \mid H \text{ is a simple unblocked path with respect to } f\}.$$

In the exceptional case where there is no simple unblocked path with respect to x, we define $D(p,f) = 0$. In this case, we must have $b_{ij} = c_{ij}$ for all (i,j) since any arc (i,j) with $b_{ij} < c_{ij}$ gives rise to a one-arc unblocked path with respect to x.

The scalar $\beta(p)$ given by

$$\beta(p) = \min\{D(p,f) \mid f \text{ is feasible}\}. \tag{37}$$

plays an important role in the subsequent analysis. The following result has been proved for the ϵ-relaxation method in [BeE87], [BeE88], and [BeT89], and is based on showing that the relation

$$p_i - p_i^0 < (N-1)\epsilon + \beta(p^0), \tag{38}$$

where p^0 is the initial price vector, holds throughout the algorithm for all nodes i with $g_i > 0$. The proof for the network auction algorithm is identical; see also the proof of Prop. 3:

Lemma 2: For every node, the total number of substantive price rises of a subset containing the node, up to termination of the network auction algorithm, is $O\big(N + \beta(p^0)/\epsilon\big)$, where p^0 is the initial price vector.

Our main complexity result is the following:

Proposition 5: Let Assumptions 1-5 hold and let p^0 be the initial price vector. Then the network auction algorithm terminates in $O\big(N^3 + N^2\beta(p^0)/\epsilon\big)$ time.

Proof: To economize on notation, we write β in place of $\beta(p^0)$. We will also need to distinguish between nonsaturating δ-pushes in Step 2 for which $\delta < r_j - g_j$ or $\delta = r_j - g_j$ in Eq. (22); these

are called *regular* and *irregular* nonsaturating δ-pushes, respectively. Note that for each irregular δ-push, the node j of Eq. (22) enters the set L and participates in a substantive price rise in the subsequent Step 3. The dominant computational requirements of the network auction algorithm are:

(1) The computation required for price rises and for updating the push lists of nodes involved in the price rises.

(2) The computation required for restructuring the list T following price rises.

(3) The computation required for saturating δ-pushes.

(4) The computation required for irregular nonsaturating δ-pushes.

(5) The computation required for regular nonsaturating δ-pushes.

(6) The computation required for selecting a node i with $g_i > 0$ in Step 0.

There is also additional computation for updating the reject capacities of various nodes, but this work can be lumped into the work for δ-pushes and price rises, with no effect on the subsequently derived complexity bound.

We will show that the times required for the operations in (1)-(6) above can be estimated as follows:

For (1), $O\big(A(N+\beta/\epsilon)\big)$.

For (2), $O\big(N^2(N+\beta/\epsilon)\big)$; if the alternative form of Step 3 is used, the time required is $O\big(A(N+\beta/\epsilon)\big)$.

For (3), $O\big(A(N+\beta/\epsilon)\big)$.

For (4), $O\big(A(N+\beta/\epsilon)\big)$.

For (5), $O\big(N^2(N+\beta/\epsilon)\big)$.

For (6), $O\big(N^2(N+\beta/\epsilon)\big)$.

Thus, we will obtain the desired $O\big(N^2(N+\beta/\epsilon)\big)$ time bound.

Indeed, since by Lemma 2, there are $O(N+\beta/\epsilon)$ price increases for each node and a total of $O\big(N(N+\beta/\epsilon)\big)$ price rises, the time required for (1) is $O\big(A(N+\beta/\epsilon)\big)$ and the time required for (2) is $O\big(N^2(N+\beta/\epsilon)\big)$. If the alternative form of Step 3 is used, then the restructuring of the list T can be done by placing sequentially the nodes of L at the top of T as their price is increased, so that the time required for (2) is $O\big(A(N+\beta/\epsilon)\big)$.

Whenever an arc flow is set to either the upper or the lower bound due to a saturating push at one of the end nodes, it takes a price increase of at least 2ϵ by the opposite end node before the arc flow can change again. Therefore, there are $O(N+\beta/\epsilon)$ saturating pushes per arc. The computation time for each of these, including the time to remove the arc from the corresponding push list, is $O(1)$,

so the time required for (3) is $O\big(A(N+\beta/\epsilon)\big)$. Similarly, for each irregular nonsaturating δ-push there is a price rise of the corresponding node j that enters the set L. Thus there are $O(N+\beta/\epsilon)$ irregular nonsaturating pushes per arc, and the time required for (4) is $O\big(A(N+\beta/\epsilon)\big)$.

There remains to estimate the computational requirements for (5) and (6). At this point, we will use the assumption that the algorithm is operated in cycles with the node order in each cycle determined by the list T. We will demonstrate that the number of cycles up to termination is $O\big(N(N+\beta/\epsilon)\big)$. Given this, we argue that for each cycle, there can be only one regular nonsaturating push per node in Step 2, for a total of $O\big(N^2(N+\beta/\epsilon)\big)$ regular nonsaturating pushes. Since the time required for each of these pushes is $O(1)$, the time required for (5) is $O\big(N^2(N+\beta/\epsilon)\big)$. Furthermore, the time to select a positive surplus node in Step 0 is $O(N)$ per cycle, so the time required for (6) is also $O\big(N^2(N+\beta/\epsilon)\big)$. Thus the proof of the time estimates for the computations (1)-(6) stated above will be completed.

To show that the number of cycles up to termination is $O\big(N(N+\beta/\epsilon)\big)$, we use the admissible graph $G^* = (N, A^*)$ and we argue as follows: a node i is called a *predecessor* of a node j if a directed path starting at i, ending at j, and having arcs oriented from i to j, exists in G^*. First, we claim that immediately following a price rise of a node set I, there are no arcs (j,i) in A^* with $j \notin I$ and $i \in I$. To see this, note that if $(j,i) \in A$ with $j \notin I$ and $i \in I$ is ϵ^+-unblocked after the price rise, we must have $p_j > p_i + a_{ji} + \epsilon$ before the price rise, and, hence, $x_{ji} = c_{ji}$, implying that (i,j) is not in A^*. The ϵ^--unblocked case is similar, establishing the claim. We next claim that G^* is always acyclic. This is true initially because, by Assumption 4, A^* is empty. δ-pushes can only remove arcs from A^*, so G^* can acquire a cycle only immediately after a price rise of a node set I, and the cycle must include nodes of I as well as some nodes not in I. But since there are no arcs (j,i) with $j \notin I$ and $i \in I$ in the admissible graph following the price rise, no such cycle is possible. This establishes the second claim. Finally, we claim that the node list T maintained by the algorithm will always be compatible with the partial order induced by G^*, in the sense that every node will always appear in the list after all its predecessors. Again this is initially true because A^* starts out empty. Furthermore a δ-push does not create new predecessor relationships, while after a price rise of a node set node I, there can be no predecessor of a node in I which does not belong to I, while the set I is moved to the top of the list before any possible descendants. This establishes the claim.

Let N^+ be the set of nodes with positive surplus that have no predecessor with positive surplus, and let N^0 be the set of nodes with nonpositive surplus that have no predecessor with positive surplus. Then, as long as no price rise takes place, all nodes in N^0 remain in N^0, and execution of an iteration starting at a node $i \in N^+$ moves i from N^+ to N^0. If there is no price rise during a cycle, then all nodes of N^+ will be added to N^0 by the end of the cycle, which implies that the algorithm terminates. Therefore, there will be a price rise during every cycle except possibly for the last one.

Since the number of price increases per node is $O(N + \beta/\epsilon)$, there can be only $O\big(N(N + \beta/\epsilon)\big)$ cycles.

The proof of the time estimates for (1)-(6) stated above is now complete and the desired overall time bound for the algorithm follows. **Q.E.D.**

Note that the classical max-flow problem can be formulated so that all arc costs a_{ij} are zero except for one arc cost which is unity ([BeT89], p. 334), and with an initial price vector p^0 such that $p^+ - p^- = O(1)$, we have $\beta(p^0) = O(1)$ [cf. Eq. (36)]. By taking $\epsilon = 1/(N+1)$ in Prop. 5, it follows that the network auction algorithm solves the problem in $O\big(N^3\big)$ time.

The Scaled Version of the Algorithm

We now describe a scaled version of the network auction algorithm. Its analysis is virtually identical to the corresponding analysis of the ϵ-relaxation method given in the sources mentioned earlier.

Consider the problem obtained from (LNF) by multiplying all arc cost coefficients by $N+1$, that is, the problem with arc cost coefficients

$$a'_{ij} = (N+1)a_{ij}, \qquad \forall\ (i,j) \in A.$$

We refer to this problem as (SNLF). If a pair (x', p') satisfies 1-CS (namely, ϵ-CS with $\epsilon = 1$) with respect to (SLNF), then clearly the pair

$$(x, p) = \left(x', \frac{p'}{N+1}\right)$$

satifies $(N+1)^{-1}$-CS with respect to (LNF), and hence x' is optimal for (LNF) by Prop. 1. In the scaled algorithm, we seek a 1-CS solution to (SLNF).

Let

$$M = \lfloor \log_2(N+1)C \rfloor + 1 = O\big(\log(NC)\big), \tag{39}$$

where $C = \max_{(i,j)\in A} |a_{ij}|$. In the scaled algorithm, we solve M subproblems. The mth subproblem is a minimum cost flow problem, where the cost coefficient of each arc (i,j) is

$$a_{ij}(m) = \text{Trunc}\left(\frac{a'_{ij}}{2^{M-m}}\right), \tag{40}$$

where Trunc$(\cdot)$ denotes integer rounding in the direction of 0, that is, down for positive and up for negative numbers. Note that $a_{ij}(m)$ is the integer consisting of the m most significant bits in the M-bit binary representation of a'_{ij}. In particular each $a_{ij}(1)$ is 0, +1, or -1, while $a_{ij}(m+1)$ is obtained

by doubling $a_{ij}(m)$ and adding (subtracting) 1 if the $(m+1)$st bit of the M-bit representation of a_{ij} is a 1 and a_{ij} is positive (negative). Note also that

$$a_{ij}(M) = a'_{ij},$$

so the last problem of the sequence is (SLNF).

All problems in the sequence are solved by applying the ϵ-relaxation algorithm using $\epsilon = 1$, yielding upon termination a pair $\big(f^t(m), p^t(m)\big)$ satisfying 1-CS with respect to the cost coefficients $a_{ij}(m)$. The algorithm is operated in cycles as per Assumption 4.

The starting pair $\big(f^0(1), p^0(1)\big)$ for the first problem must be integer and must satisfy 1-CS. The starting price vector for the $(m+1)$st problem $(m = 1, 2, ..., M-1)$ is

$$p^0(m+1) = 2p^t(m), \tag{41}$$

where $p^t(m)$ is the final price vector obtained from solution of the mth problem. Doubling $p^t(m)$ as above roughly maintains complementary slackness since $a_{ij}(m)$ is roughly doubled when passing to the $(m+1)$st problem. Indeed, it can be seen that every arc that was 1-balanced (1-active, 1-inactive) upon termination of the algorithm for the mth problem will be 3-balanced (1-active, 1-inactive, respectively) at the start of the $(m+1)$st problem.

The starting flow vector $x^0(m+1)$ for the $(m+1)$st problem is obtained from $x^t(m)$ by setting

$$x^0_{ij}(m+1) = x^t_{ij}(m) \qquad \text{if } -\epsilon < p_i - p_j - a_{ij} < \epsilon,$$

$$x^0_{ij}(m+1) = c_{ij} \qquad \text{if } p_i - p_j - a_{ij} \geq \epsilon,$$

$$x^0_{ij}(m+1) = b_{ij} \qquad \text{if } -\epsilon \geq p_i - p_j - a_{ij}.$$

Note that this initialization method implies that the starting price and flow vector will be integer, and that there will be no 1^+-unblocked and 1^--unblocked arcs initially for the $(m+1)$st problem. These facts guarantee that Assumptions 2, 3, and 4 are satisfied for the subproblems.

Based on Prop. 5, the scaled form of the algorithm solves the problem in $O(MN^3 + N^2B)$ time, where

$$B = \sum_{m=1}^{M} \beta_m\big[p^0(m)\big], \tag{42}$$

and $\beta_m(\cdot)$ is defined by Eq. (37) but with the modified cost coefficients $a_{ij}(m)$ replacing a_{ij} in the definition (35). We will need the following result, which is shown in [BeE87], [BeE88], and [BeT89]:

Lemma 3: If there exists a feasible flow vector x satisfying γ-CS together with p for some $\gamma \geq 0$, then

$$0 \leq \beta(p) \leq (N-1)\gamma.$$

We will show that $\beta_m(p^0(m)) = O(N)$ for every m, and we will then use Lemma 4 to obtain the following result, which is identical to the one shown in [BeE87], [BeE88], and [BeT89] for the ϵ-relaxation method:

Proposition 6: Assume that for the initial subproblem, Assumptions 1-5 are satisfied and that $p_i^0 - p_j^0 = O(1)$ for all arcs (i,j). The scaled form of the algorithm solves the problem in $O(N^3 \log(NC))$ time, where $C = \max_{(i,j)\in A} |a_{ij}|$.

Proof: Since initially we have $p_i - p_j = O(1)$ and $a_{ij}(1) = O(1)$ for all arcs (i,j), we obtain $d_H(p^0(1)) = O(N)$ for all H, and $\beta_1(p^0(1)) = O(N)$. We also have that the final flow vector $x^t(m)$ obtained from the mth problem is feasible, and together with $p^0(m+1)$ it can be easily seen to satisfy 3-CS. It follows from Lemma 3 that $\beta_{m+1}(p^0(m+1)) \leq 3(N-1) = O(N)$ and the result follows from Eq. (42) as discussed above. **Q.E.D.**

Problems with Unit Arc Capacities

When the feasible flow range of each arc is $[0,1]$, such as for example the assignment problem and the k node-disjoint shortest path problem, there are no regular nonsaturating pushes. For this reason, to obtain a good complexity bound, it is not necessary to maintain and restructure the list T as described earlier. Instead, a much simpler FIFO queue that includes the nodes with positive surplus can be used. With this algorithmic modification, the preceding analysis can be adapted to show that the complexity bound is reduced to $O(A(N + \beta(p_0)/\epsilon))$. When scaling is used, we can then obtain with a similar analysis an $O(NA\log(NC))$ bound.

Complexity of the Generic Algorithm

Much of the preceding complexity analysis can also be applied to the generic algorithm under some broadly applicable assumptions. In particular, let us call a δ-push by node i *exhaustive* on arc (i,j) [or arc (j,i)] if $\delta = \min\{g_i, c_{ij} - x_{ij}\}$ [or $\delta = \min\{g_i, x_{ji} - b_{ji}\}$, respectively]. Let also n_i be the number of times that the price of node i is changed due to a price rise. Consider in addition to Assumptions 1-3 of the previous section, the following assumptions:

(a) The computation required for price rises is bounded by a constant times $\sum_{i\in\mathcal{N}} a_i n_i$, where a_i is the number of incident arcs of node i.

(b) Each δ-push requires $O(1)$ computation and the number of δ-pushes which are not exhaustive

is bounded by a constant times $\sum_{i \in \mathcal{N}} n_i$.

(c) Between two successive price rises there can be at most N^2 exhaustive δ-pushes. (This assumption in particular is satisfied is satisfied in the network auction algorithm if the node selection policy is arbitrary but the algorithm is operated so that the admissible graph is acyclic.)

Under these assumptions our earlier analysis can be easily modified to show an $O\big(N^4 \log(NC)\big)$ running time for the scaled version of the generic algorithm. In particular, for fixed ϵ, by using assumptions (a) and (b) above, we can show similar to the proof of Prop. 5 that the computation for price rises, saturating δ-pushes, and nonexhaustive δ-pushes is $O\big(A(N + \beta/\epsilon)\big)$. By using assumptions (a) and (c) above we can also show similar to the proof of Prop. 5 that the computation for nonsaturating δ-pushes is $O\big(N^3(N + \beta/\epsilon)\big)$. We thus obtain a $O\big(N^3(N + \beta/\epsilon)\big)$ bound for applying the algorithm with a single fixed ϵ, which with scaling translates to a $O\big(N^4 \log(NC)\big)$ overall bound. By exploiting the problem structure and by using data structures such as the ones of the sweep implementation, it may be possible to reduce the time bound for nonsaturating δ-pushes, which is the worst-case complexity bottleneck. Such data structures can be developed in the context of particular algorithms, e.g. the network auction algorithm. Note that for the case of unit arc capacities, there are no nonsaturating δ-pushes and we obtain the earlier $O\big(NA \log(NC)\big)$ bound.

6. COMPUTATIONAL RESULTS

In this section we present the results of some of our experimentation with various implementations of special cases of the generic algorithm. In particular, we will present results using the transportation auction algorithm of Section 3 and using the k node-disjoint shortest path algorithm of Section 4. The reader is also referred to several computational studies that have tested extensively auction algorithms for assignment problems [BeC89b], [BeC89c], [Ber90], [KKZ89], [PhZ88], [WeZ90], [WeZ91], [Zak90].

Transportation Problems

We compared our auction code for solving transportation problems, called TRANSAUCTION, against the state-of-the-art minimum cost flow code RELAX-II, described in [BeT85], [BeT88]. In [BeC89a] we also compared TRANSAUCTION with RELAX-II, as well as with the primal simplex

code RNET in, but the experimentation reported here was done with much larger problems. Our computational results, summarized in Figs. 5 and 6, generally agree with the conclusions of [BeC89a]. In particular, while TRANSAUCTION is not uniformly superior to RELAX-II, it runs much faster for important classes of tranportation problems. Generally these problems are characterized by two properties, which we call *homogeneity* and *asymmetry*. A homogeneous problem is one for which there are only few levels of supply and demand. An asymmetric problem is one for which the number of sources is much larger than the number of sinks. Based on our experience, homogeneous and asymmetric problems arise in many types of applications. As Fig. 6 shows, TRANSAUCTION has an advantage even for problems that are not homogeneous but are, however, very asymmetric.

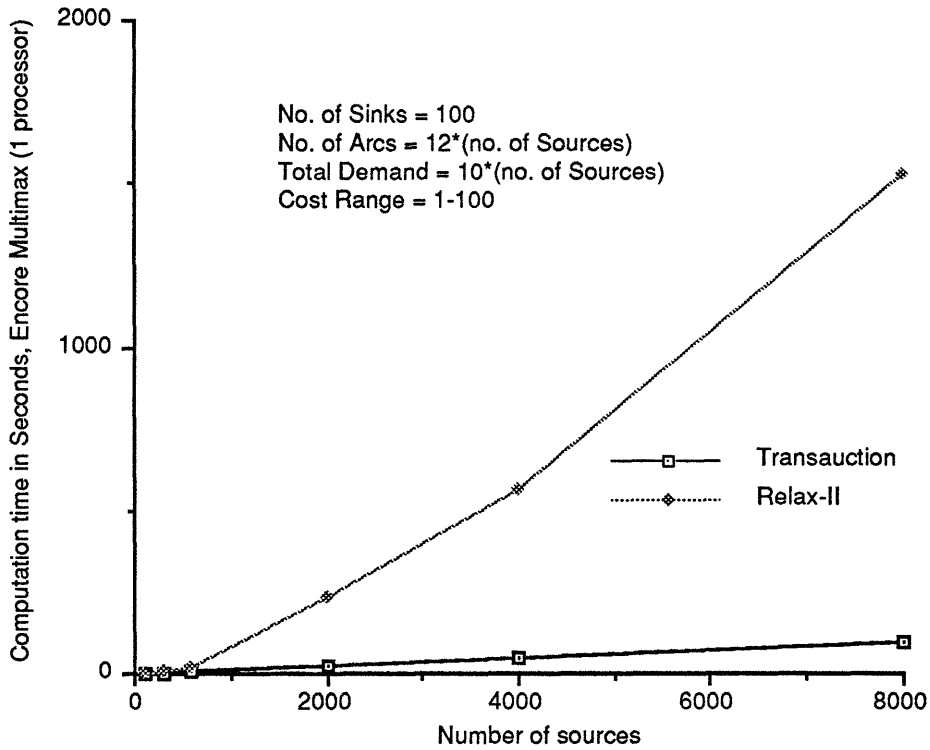

Figure 5 Comparison of TRANSAUCTION and RELAX-II for a constant number of sinks and average node degree, as the number of sources increases. The relative advantage of TRANSAUCTION exceeds an order of magnitude as N increases.

k Node-Disjoint Shortest Path Problems

We have implemented the auction algorithm for k node-disjoint shortest path problems in a code called AUCTION-KSP, which we tested against an implementation of the ϵ-relaxation method, called E-RELAX (given in [Ber91]), the RELAXT-III code, (a more recent version of the RELAX-II code

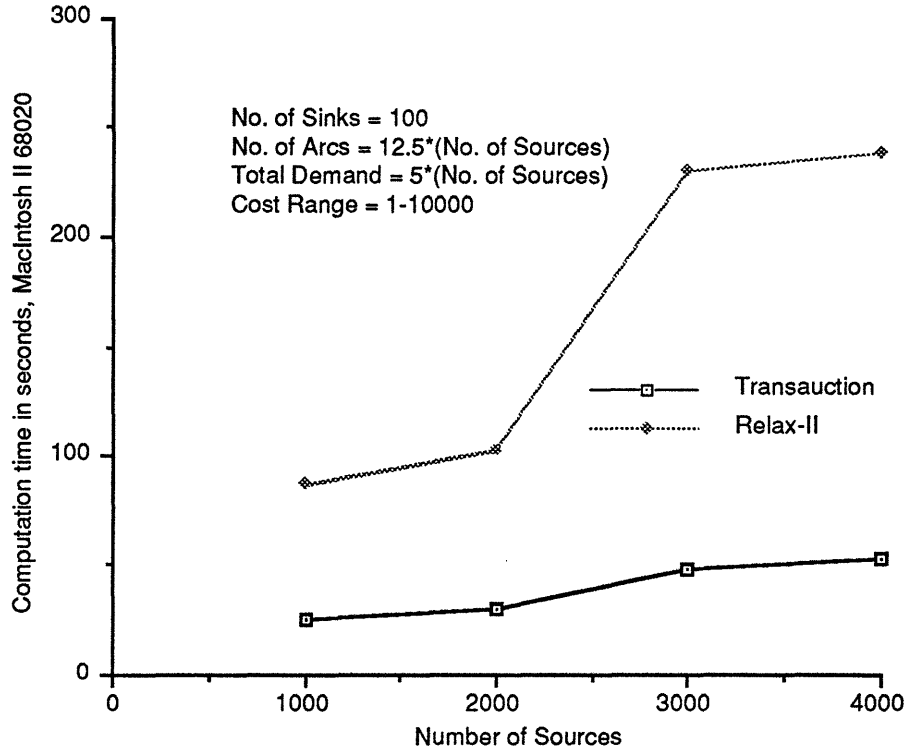

Figure 6 Comparison of TRANSAUCTION and RELAX-II for asymmetric problems, generated with the problem generator NETGEN [KNS74]. Here the supplies and demands of all sources and sinks, respectively, are randomly generated using a uniform distribution, resulting in inhomogeneous problems. TRANSAUCTION still holds a significant advantage.

used for the results of Figs. 5 and 6), which is described in [BeT90], and the primal-simplex code NETFLO, which is given in [KeH80]. Figures 7 and 8 give some representative experimental results. NETFLO was slower by an order of magnitude than RELAXT-III and E-RELAX for the problems we tried, so its performance is not shown in these figures. AUCTION-KSP does not use scaling and this probably slows down its performance, particularly when k is relatively large. Despite this fact, AUCTION-KSP is uniformly and substantially faster that RELAXT-III and much faster than E-RELAX. This suggests that our specialized auction algorithm for the k node-disjoint shortest path problem is not just a heuristic improvement on the ϵ-relaxation method, but rather embodies some computational ideas that are genuinely interesting. We note also that the performance of AUCTION-KSP will probably improve substantially once we use scaling as well as "down iterations" where the prices of nodes with negative surplus are decreased. Down iterations have been shown to be very useful in the context of reverse auction for assignment problems [BCT91], [Ber91a], and reverse auction for shortest path problems [Ber91a], [Ber91b].

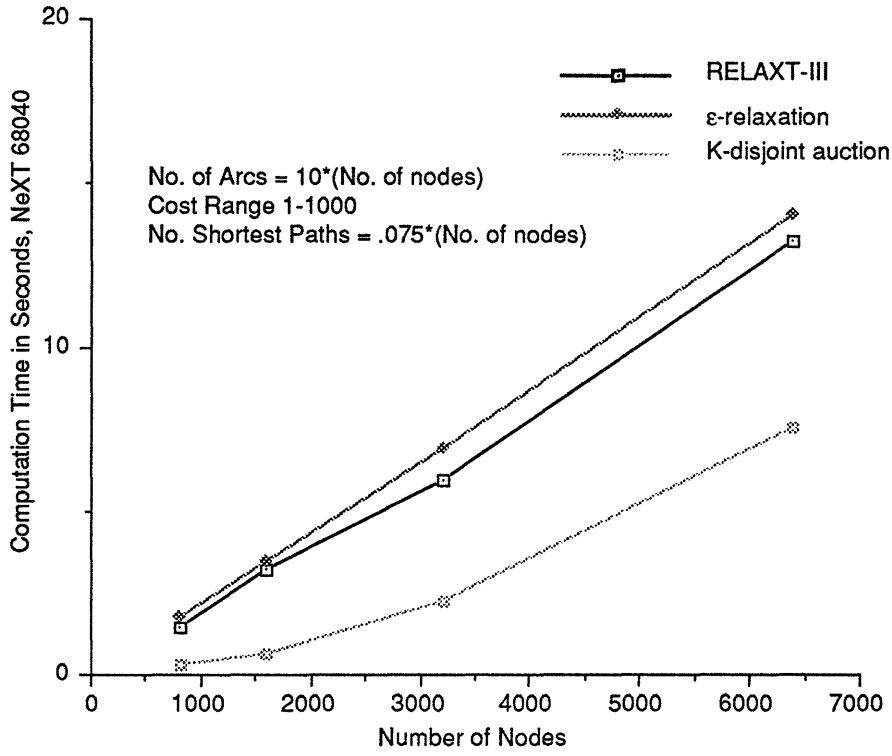

Figure 7 Comparison of the auction code AUCTION-KSP for k node-disjoint shortest path problems, with the transhipment codes E-RELAX, and RELAXT-III. Here the problems have a constant k while the number of nodes increases. The graphs of these problems were generated using NETGEN.

Additional Comments

We have also conducted much additional experimentation with the purpose to determine for what types of problems auction-like algorithms can form the basis for codes that outperform current state-of-the-art codes. This experimentation is not conclusive and cannot be presented here. However, the results seem to suggest that problems with a structure resembling the one of the assignment problem (bipartite or nearly bipartite structure, small and/or uniform sized supplies, small arc capacities) are good candidates for effective solution using specialized versions of the generic auction algorithm. Also a relatively simple problem structure such as the one of the max-flow, shortest path, and other related problems seems to favor the use of specialized auction algorithms. For general linear transhipment problems without a particular special structure we found that our best implementations of the ϵ-relaxation and network auction algorithms were considerably slower (by a factor of the order of three to ten) than state-of-the-art relaxation codes.

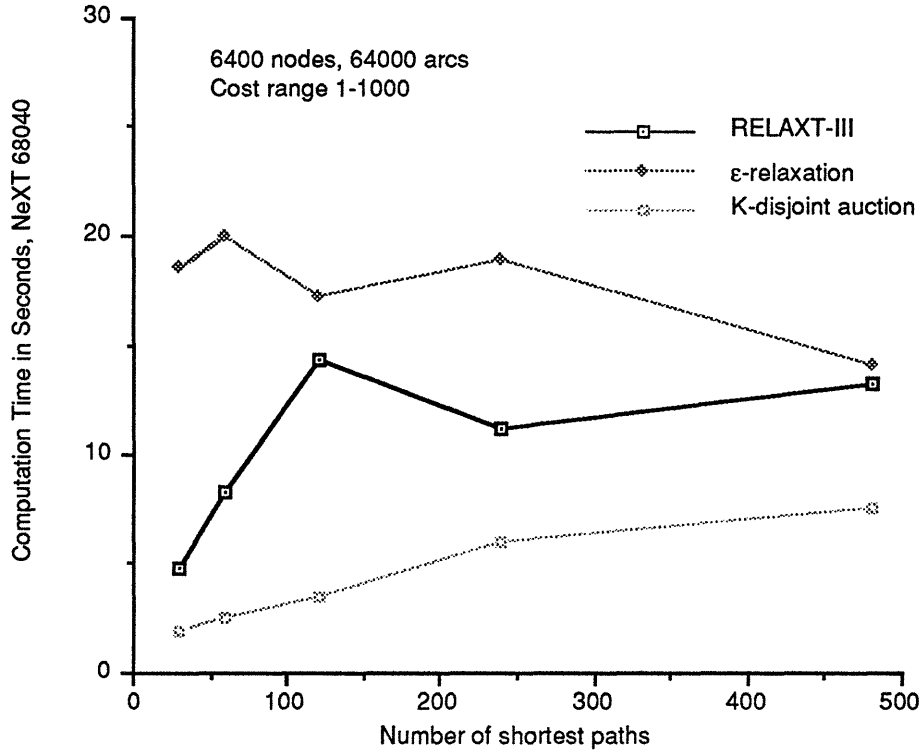

Figure 8 Comparison of the auction code AUCTION-KSP for k node-disjoint shortest path problems, with the transhipment codes E-RELAX, and RELAXT-III. Here there is a single graph that has 6400 nodes and 64000 arcs, but the number of shortest paths k varies.

REFERENCES

[BCT91] Bertsekas, D. P., Castañon, D. A., and Tsaknakis, H., "Reverse Auction and the Solution of Inequality Constrained Assignment Problems," Alphatech, Inc. Report, March 1991.

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

[BeC89a] Bertsekas, D. P., and Castañon, D. A., "The Auction Algorithm for Transportation Problems," Annals of Operations Research, Vol. 20, 1989, pp. 67-96.

[BeC89b] Bertsekas, D. P., and Castañon, D. A., "The Auction Algorithm for the Minimum Cost Network Flow Problem," Laboratory for Information and Decision Systems Report LIDS-P-1925, M.I.T., Cambridge, MA, November 1989.

[BeC89c] Bertsekas, D. P., and Castañon, D. A., "Parallel Synchronous and Asynchronous Imple-

mentations of the Auction Algorithm," Alphatech Report, Burlington, MA, Nov. 1989; also Parallel Computing, Vol. 17, 1991, pp. 707-732.

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

[BeE88] Bertsekas, D. P., and Eckstein, J., "Dual Coordinate Step Methods for Linear Network Flow Problems," Laboratory for Information and Decision Systems Report LIDS-P-1768, M.I.T., Cambridge, MA, 1988, also in Math. Progr., Series B, Vol. 42, 1988, pp. 203-243.

[BeM73] Bertsekas, D. P., and Mitter, S. K., "A Descent Numerical Method for Optimization Problems with Nondifferentiable Cost Functionals," SIAM J. on Control, Vol. 11, 1973, pp. 637-652.

[BeT85] Bertsekas, D. P., and Tseng, P., "Relaxation Methods for Minimum Cost Ordinary and Generalized Network Flow Problems," LIDS Report P-1462, M.I.T., May 1985; also Operations Research J., Vol. 36, 1988, pp. 93-114.

[BeT88] Bertsekas, D. P., and Tseng, P., "RELAX: A Computer Code for Minimum Cost Network Flow Problems," Annals of Operations Research, Vol. 13, 1988, pp. 127-190.

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

[Ber79] Bertsekas, D. P., "A Distributed Algorithm for the Assignment Problem," Lab. for Information and Decision Systems Working Paper, M.I.T., March 1979.

[Ber82] Bertsekas, D. P., "A Unified Framework for Minimum Cost Network Flow Problems," Laboratory for Information and Decision Systems Report LIDS-P-1245-A, M.I.T., Cambridge, MA, 1982; also in Math. Programming, 1985, pp. 125-145.

[Ber86a] Bertsekas, D. P., "Distributed Asynchronous Relaxation Methods for Linear Network Flow Problems," LIDS Report P-1606, M.I.T., Nov. 1986.

[Ber86b] Bertsekas, D. P., "Distributed Relaxation Methods for Linear Network Flow Problems," Proceedings of 25th IEEE Conference on Decision and Control, Athens, Greece, 1986, pp. 2101-2106.

[Ber88] Bertsekas, D. P., "The Auction Algorithm: A Distributed Relaxation Method for the Assignment Problem," Annals of Operations Research, Vol. 14, 1988, pp. 105-123.

[Ber90] Bertsekas, D. P., "The Auction Algorithm for Assignment and Other Network Flow Problems: A Tutorial," Interfaces, Vol. 20, 1990, pp. 133-149.

[Ber91a] Bertsekas, D. P., Linear Network Optimization: Algorithms and Codes, M.I.T. Press, Cambridge, MA., 1991.

[Ber91b] Bertsekas, D. P., "The Auction Algorithm for Shortest Paths," SIAM J. on Optimization, Vol. 1, 1991, pp. 425-447.

[Dre69] Dreyfus, S. E., "An Appraisal of Some Shortest-Path Algorithms," Operations Research, Vol. 17, 1969, pp. 395-412.

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

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

[KNS74] Klingman, D., Napier, A., and Stutz, "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.

[KKZ89] Kempa, D., Kennington, J., and Zaki, H., "Performance Characteristics of the Jacobi and Gauss-Seidel Versions of the Auction Algorithm on the Alliant FX/8," Report OR-89-008, Dept. of Mech. and Ind. Eng., Univ. of Illinois, Champaign-Urbana, 1989.

[KeH80] Kennington, J., and Helgason, R., Algorithms for Network Programming, Wiley, N. Y., 1980.

[LiZ91] Li, X., and Zenios, S. A., "Data Parallel Solutions of Min-Cost Network Flow Problems Using ϵ-Relaxations," Report 1991-05-20, Dept. of Decision Sciences, The Wharton School, Univ. of Pennsylvania, Phil., Penn.

[PaS82] Papadimitriou, C. H., and Steiglitz, K., Combinatorial Optimization: Algorithms and Complexity, Prentice-Hall, Englewood Cliffs, N. J., 1982.

[PhZ88] Phillips, C., and Zenios, S. A., 1988. "Experiences with Large Scale Network Optimization on the Connection Machine," Report 88-11-05, Dept. of Decision Sciences, The Wharton School, Univ. of Pennsylvania, Phil., Penn.; also in "The Impact of Recent Computing Advances on Operations Research," Elsevier Publishing Co., Vol. 9, pp. 90-99, 1990.

[Roc80] Rock, H., "Scaling Techniques for Minimal Cost Network Flows," in Discrete Structures and Algorithms, by V. Page (ed.), Carl Hansen, Munich, 1980.

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

[TsB87a] Tseng, P., and Bertsekas, D. P., "Relaxation Methods for Linear Programs," Mathematics of Operations Research, Vol. 12, 1987, pp. 569-596.

[TsB87b] Tseng, P., and Bertsekas, D. P., "Relaxation Methods for Monotropic Programs," Laboratory for Information and Decision Systems Report LIDS-P-1697, M.I.T., Cambridge, Mass., Aug. 1987; also Math. Programming, Vol. 46, 1990, pp. 127-151.

[WeZ90] Wein, J., and Zenios, S. A., "Massively Parallel Auction Algorithms for the Assignment Problem," Proc. of 3rd Symposium on the Frontiers of Massively Parallel Computation, Md.

[WeZ91] Wein, J., and Zenios, S. A., "On the Massively Parallel Solution of the Assignment Problem," J. of Parallel and Distributed Computing, Vol. 13, 1991, pp. 228-236.

[Zak90] Zaki, H., "A Comparison of Two Algorithms for the Assignment Problem," Report ORL 90-002, Dept. of Mechanical and Industrial Engineering, Univ. of Illinois, Urbana, Ill.