AN AUCTION/SEQUENTIAL SHORTEST PATH ALGORITHM FOR THE MINIMUM COST NETWORK FLOW PROBLEM[1]

by

Dimitri P. Bertsekas[2]

Abstract

We propose a new algorithm for the solution of the linear minimum cost network flow problem, based on a sequential shortest path augmentation approach. Each shortest path is constructed by means of the recently proposed auction/shortest path algorithm. This approach allows useful information to be passed from one shortest path construction to the next. However, the naive implementation of this approach where the length of each arc is equal to its reduced cost fails because of the presence of zero cost cycles. We overcome this difficulty by using as arc lengths ϵ-perturbations of reduced costs and by using ϵ-complementary slackness conditions in place of the usual complementary slackness conditions. We present several variants of the main algorithm, including one that has proved very efficient for the max-flow problem. We also discuss the possibility of combining the algorithm with the relaxation method and we provide encouraging computational results.

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

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

1. INTRODUCTION

We consider the classical minimum cost network flow problem, involving a directed graph with node set $\mathcal{N}$ and arc set $\mathcal{A}$. The number of nodes is denoted by N and number of arcs is denoted by 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

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

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 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 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. A flow vector satisfying the constraints (1) and (2) is called *feasible*. For a given flow vector x, the *surplus* of node i is defined as the difference between the supply of i and the net outflow from i,

$$g_i = s_i + \sum_{\{j|(j,i)\in\mathcal{A}\}} x_{ji} - \sum_{\{j|(i,j)\in\mathcal{A}\}} x_{ij}. \tag{3}$$

In this paper, we propose a new algorithm that combines ideas from the Ford-Fulkerson primal-dual (or sequential shortest path) method [FoF57], [FoF62], and the author's ϵ-relaxation [Ber86a], [Ber86b], and auction/shortest path methods [Ber91b] (see e.g. the textbooks [BeT89] and [Ber91a] for detailed descriptions and analysis of these methods). The algorithm consists of a sequence of augmentations along shortest paths, which are constructed by means of the auction/shortest path method. However, the method does not work when the arc lengths are equal to the reduced costs of the arcs as in the usual Ford-Fulkerson method. It is necessary to perturb by ϵ the reduced costs and to use ϵ-complementary slackness in place of the usual complementary slackness.

The paper is organized as follows. Section 2 describes the main algorithm and shows its finite termination to an optimal flow vector when the problem is feasible. Several variants of the main algorithm together with implementation issues are discussed in Section 3. An important variation uses augmenting paths that are not necessarily shortest. This variant is the basis for a very efficient max-flow algorithm proposed and analyzed in [Ber93]. In Section 4 we provide computational results of various versions of the algorithm, including combinations with the relaxation algorithm of [Ber85] and the RELAX code of [BeT88a], [BeT88b]. These results suggest that the algorithm of this paper outperforms the ϵ-relaxation method, and when combined with the relaxation method, it improves dramatically its performance on the types of problems where the relaxation method may encounter slow convergence.

2. THE ALGORITHM

We introduce a price p_i for each node i, which may be viewed as a dual variable. We denote by p the vector with elements p_i, $i \in \mathcal{N}$. Given $\epsilon \geq 0$, a price vector p, and a flow vector x are said to satisfy *ϵ-complementary slackness* (ϵ-CS for short) if x satisfies the capacity constraints (2) and

$$x_{ij} < c_{ij} \qquad \Rightarrow \qquad p_i \leq a_{ij} + p_j + \epsilon \qquad \forall\ (i,j) \in \mathcal{A}, \tag{4}$$

$$b_{ji} < x_{ji} \qquad \Rightarrow \qquad p_i \leq p_j - a_{ji} + \epsilon \qquad \forall\ (j,i) \in \mathcal{A}. \tag{5}$$

ϵ-CS was introduced in [Ber86a], [Ber86b], in the context of the ϵ-relaxation method, and its utility is due in large measure to the following proposition, which relies on the integrality of the problem data (see e.g. [BeT89] or [Ber91a] for a proof).

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

Note that when $\epsilon = 0$ the ϵ-CS conditions (4) and (5) reduce to the usual complementary slackness conditions. A standard duality result (see e.g., [BeT89], [Ber91a], [PaS82], [Roc84]) states that if x is feasible and together with some p satisfies these complementary slackness conditions, then x is optimal and p is optimal for an associated dual problem.

The classical primal-dual (or sequential shortest path) method maintains a pair (x,p) satisfying complementary slackness ($\epsilon = 0$), and at each iteration constructs a shortest path from some node with positive surplus to the set of nodes with negative surplus, along which it performs an augmentation of the current flow vector. The shortest path computation is performed in the *reduced graph* $G_R = \big(\mathcal{N}, \mathcal{A}_{\mathcal{R}}\big)$ whose arc set $\mathcal{A}_{\mathcal{R}}$ consists of an arc (i,j) for each arc $(i,j) \in \mathcal{A}$ with $x_{ij} < c_{ij}$, and an arc (j,i) for each arc $(i,j) \in \mathcal{A}$ with $b_{ij} < x_{ij}$. The arc lengths are $a_{ij} + p_j - p_i$ for the arcs $(i,j) \in \mathcal{A}$ with $x_{ij} < c_{ij}$, and $p_i - a_{ij} - p_j$ for the arcs (j,i) corresponding to arcs $(i, j) \in \mathcal{A}$ with $b_{ij} < x_{ij}$. It is in principle possible to solve the shortest path problem by any shortest path method that requires nonnegative arc lengths, such as a Dijkstra-like method. The recently proposed auction algorithm for shortest paths (see [Ber91a], [Ber91b]) offers some advantages in this respect because of its ability to transfer information from one shortest path computation to the next, but requires that all cycles have strictly positive length. This method maintains a path, which is extended or contracted by a single arc at each iteration. Unfortunately, however, the method cannot be used conveniently in the context of the sequential shortest path method because the reduced graph has cycles with zero length [each arc (i, j) with $b_{ij} < x_{ij} < c_{ij}$ gives rise to the zero length arcs (i, j) and (j, i) in the reduced graph], and the path maintained by the auction/shortest path method can "double up on itself" and close a cycle.

To overcome this difficulty, it is possible to use auction/shortest path algorithms with graph reduction, as proposed in [PaS91] and [BPS92]. However, this requires considerable overhead and a separation of the shortest path construction process from the price change operations of the primal-dual algorithm. In this paper we use an alternative approach, where the auction/shortest path construction process is blended harmoniously with the remainder of the algorithm. In this approach, we use ϵ-perturbations of the arc lengths, which ensure that the path generated by the auction/shortest path method does not close a cycle through an extension. We first introduce some terminology.

Given a flow-price pair (x, p) satisfying ϵ-CS, an arc (i, j) is said to be ϵ^+-*unblocked* if

$$p_i = p_j + a_{ij} + \epsilon \qquad \text{and} \qquad x_{ij} < c_{ij}, \tag{6}$$

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

$$p_i = p_j - a_{ji} + \epsilon \qquad \text{and} \qquad b_{ji} < x_{ji}. \tag{7}$$

The *admissible graph* corresponding to (x, p) is defined as $G^* = (\mathcal{N}, \mathcal{A}^*)$, where the arc set $\mathcal{A}^*$ consists of an arc (i, j) for each ϵ^+-unblocked arc $(i, j) \in \mathcal{A}$, and an arc (i, j) for each ϵ^--unblocked arc $(j, i) \in \mathcal{A}$.

A path P is a sequence of nodes $(n_1, n_2, \ldots, n_k)$ and a corresponding sequence of $k-1$ arcs such that the ith arc in the sequence is either (n_i, n_{i+1}) (in which case it is called a *forward* arc) or (n_{i+1}, n_i) (in which case it is called a *backward* arc). For any path P, we denote by $s(P)$ and $t(P)$ the start and terminal nodes of P, respectively, and by P^+ and P^- the sets of forward and backward arcs of P, respectively. We say that the path is *simple* if it has no repeated nodes. The path P is said to be ϵ-*unblocked* if all arcs of P^+ are ϵ^+-unblocked, and all arcs of P^- are ϵ^--unblocked. If P is ϵ-unblocked and the start node $s(P)$ has positive surplus and the terminal node $t(P)$ has negative surplus, we say that P is an *augmenting path*. An *augmentation* along such a path consists of increasing the flow of all arcs in P^+ and reducing the flow of all arcs in P^- by the common increment

$$\delta = \min\left\{ g_{s(P)},\ -g_{t(P)},\ \min_{(i,j)\in P^+}\{c_{ij} - x_{ij}\},\ \min_{(i,j)\in P^-}\{x_{ij} - b_{ij}\} \right\}. \tag{8}$$

Given a path $P = (n_1, n_2, \ldots, n_k)$, a *contraction* of P is the operation that deletes the terminal node of P together with the corresponding terminal arc. An *extension* of P by an arc (n_k, n_{k+1}) or an arc (n_{k+1}, n_k), replaces P by the path $(n_1, n_2, \ldots, n_k, n_{k+1})$ and adds to P the corresponding arc. For convenience of expression we allow a path P to consist of a single node i, in which case extension by an arc (i, j) or (j, i) gives a path with start node i and terminal node j.

The algorithm of this paper uses a fixed $\epsilon > 0$, and maintains a flow-price pair (x, p) satisfying ϵ-CS and also a simple path P (possibly consisting of a single node). It terminates when all nodes have nonnegative surplus; then either all nodes have zero surplus and x is feasible, or else some node has negative surplus showing that the problem is infeasible. Throughout the algorithm, x is integer, and (x, p) and P satisfy the following:

(a) The admissible graph corresponding to (x, p) is acyclic.

(b) P belongs to the admissible graph, i.e., it is ϵ-unblocked. Furthermore, P starts at a node with positive surplus, and all its nodes have nonnegative surplus.

We assume that at the start of the algorithm we have a pair (x, p) satisfying ϵ-CS, as well as the above two properties. In particular, initially one may choose any price vector p, select x according to

$$x_{ij} = \begin{cases} c_{ij} & \text{if } p_i \geq a_{ij} + p_j, \\ b_{ij} & \text{if } p_i < a_{ij} + p_j, \end{cases} \tag{9}$$

and choose P to consist of a single node with positive surplus. For these choices, ϵ-CS is satisfied and the corresponding admissible graph is acyclic, since its arc set is empty.

At each iteration, the path P is either extended or contracted. In the case of a contraction, the price of the terminal node of P is strictly increased. In the case of an extension, no price change occurs, but if the new terminal node has negative surplus, P becomes augmenting, and an augmentation along P is performed. Then the path P is replaced by the degenerate path that consists of a single node with positive surplus, and the process is repeated.

Typical Iteration of the Auction/Sequential Shortest Path Algorithm

Let i be the terminal node of P. If

$$p_i < \min\left\{ \min_{\{(i,j)\in\mathcal{A} \mid x_{ij} < c_{ij}\}} \{a_{ij} + p_j + \epsilon\},\ \min_{\{(j,i)\in\mathcal{A} \mid b_{ji} < x_{ji}\}} \{p_j - a_{ji} + \epsilon\} \right\}, \tag{10}$$

go to Step 1; else go to Step 2.

Step 1 (Contract path): Set

$$p_i := \min\left\{ \min_{\{(i,j)\in\mathcal{A} \mid x_{ij}<c_{ij}\}} \{a_{ij}+p_j+\epsilon\}, \min_{\{(j,i)\in\mathcal{A} \mid b_{ji}<x_{ji}\}} \{p_j-a_{ji}+\epsilon\} \right\}, \tag{11}$$

and if $i \neq s(P)$, contract P. Go to the next iteration.

Step 2 (Extend path): Extend P by an arc (i, j_i) or an arc (j_i, i) that attains the minimum in Eq. (10). If the surplus of j_i is negative go to Step 3; otherwise, go to the next iteration.

Step 3 (Augmentation): Perform an augmentation along P. If all nodes have nonpositive surplus, terminate the algorithm; otherwise, replace P by a path that consists of a single node with positive surplus and go to the next iteration.

The following proposition establishes that some basic properties are maintained by the algorithm.

Proposition 2: Suppose that at the start of an iteration:

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

(b) P belongs to the admissible graph, starts at a node with positive surplus, and all its nodes have nonnegative surplus.

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

Proof: Suppose the iteration involves a contraction. Then it can be seen that the price increase (11) preserves the ϵ-CS conditions (4) and (5). Furthermore, since only the price of node i changes and no arc flow changes, the admissible graph remains unchanged except for the incident arcs of node i. In particular, all the incident arcs of i in the admissible graph at the start of the iteration are deleted and the arcs of the admissible graph corresponding to the arcs (i, j) and (j, i) that attain the minimum in Eq. (11) are added. Since all these arcs are outgoing from i in the admissible graph, a cycle cannot be closed. Finally, following a contraction, P does not contain the terminal node i, so it belongs to the admissible graph that we had before the iteration. Thus P consists of arcs that belong to the admissible graph that we obtain after the iteration.

Suppose the iteration involves an extension. Then by the ϵ-CS conditions (4) and (5), we must have

$$p_i = \min\left\{ \min_{\{(i,j)\in\mathcal{A} \mid x_{ij}<c_{ij}\}} \{a_{ij}+p_j+\epsilon\}, \min_{\{(j,i)\in\mathcal{A} \mid b_{ji}<x_{ji}\}} \{p_j-a_{ji}+\epsilon\} \right\}, \tag{12}$$

at the start of the iteration. It follows that the path P obtained by extension is simple and ϵ-unblocked, since the extension arc (i, j_i) must belong to the admissible graph. Since no price or flow changes with an extension, the ϵ-CS conditions and the admissible graph stay unchanged following the extension. If there is a subsequent augmentation at Step 3 because the new terminal node j_i has negative surplus, the ϵ-CS conditions will not be affected, while the admissible graph will not gain any new arcs, so it will remain acyclic. **Q.E.D.**

Note that if we were to take $\epsilon = 0$ (rather than $\epsilon > 0$), the preceding proof would break down, because we would not be able to prove that the admissible graph remains acyclic following an augmentation. In particular, if following an augmentation, the flow of some arc (i, j) lies strictly between its lower and upper bound, the arc (i, j) and the arc (j, i) would both belong to the admissible graph, each with zero length, thereby closing a zero length cycle.

A sequence of iterations between two successive augmentations (or the sequence of iterations up to the first augmentation) will be called an *augmentation cycle*. Let us fix an augmentation cycle and let $\overline{p}$ be the price vector at the start of the cycle. The reduced graph $G_R = (\mathcal{N}, \mathcal{A}_\mathcal{R})$ defined earlier will not change in the course of this augmentation cycle, since no arc flow will change during the cycle, except for the augmentation at the end. Suppose that we take as arc lengths of the reduced graph the reduced costs at the start of the cycle plus ϵ. In particular, during the cycle, the arc set $\mathcal{A}_\mathcal{R}$ consists of an arc (i, j) with length $a_{ij} + \overline{p}_j - \overline{p}_i + \epsilon$ for each arc $(i, j) \in \mathcal{A}$ with $x_{ij} < c_{ij}$, and an arc (j, i) with length $\overline{p}_i - a_{ij} - \overline{p}_j + \epsilon$ for each arc $(i, j) \in \mathcal{A}$ with $b_{ij} < x_{ij}$. Note that, because $(x, \overline{p})$ satisfies ϵ-CS, the arc lengths of the reduced graph are nonnegative. However, the reduced graph does not contain zero length cycles, since any such cycle must belong to the admissible graph, which is acyclic.

Using these observations, it can now be seen that the augmentation cycle is just the auction/shortest path algorithm of [Ber91a], [Ber91b] applied to the problem of finding a shortest path from the starting node $s(P)$ to some node with negative surplus in the reduced graph G_R, using the preceding ϵ-perturbed arc lengths. To understand this, one should view $p_i - \overline{p}_i$ during the augmentation cycle as the price of node i that is maintained by the auction/shortest path algorithm. The price increments $p_i - \overline{p}_i$ obtained by the auction/shortest path algorithm are added in effect to the starting prices $\overline{p}_i$ at the end of the augmentation cycle to form the new prices that will be used for the shortest path construction of the next augmentation cycle.

By the theory of the auction/shortest path algorithm, a shortest path in the reduced graph will be found in a finite number of iterations if there exists at least one path from the starting node $s(P)$ to some node with negative surplus. Such a path is guaranteed to exist if the minimum cost flow problem (LNF) is feasible. Since the augmentation will change all the flows of the final path P by a positive integer amount, we see that each augmentation cycle reduces the total absolute surplus $\sum_{i \in \mathcal{N}} |g_i|$ by a positive integer. Therefore, there can be only a finite number of augmentation cycles, and we have shown the following proposition.

Proposition 3: Assume that the minimum cost flow problem (LNF) is feasible. Then the auction/sequential shortest path algorithm terminates with a pair (x, p) satisfying ϵ-CS. The flow vector x is feasible and is optimal if $\epsilon < 1/N$.

ϵ-Scaling

As in all auction algorithms, the practical performance of the algorithm may be degraded by "price wars", that is, prolonged sequences of iterations involving small price increases. There is a built-in potential for price wars here because with a small ϵ, the reduced graph may contain cycles with small length, which slow down the underlying auction/shortest path algorithm. (There is a cycle of length 2ϵ for every arc whose flow lies strictly between the corresponding flow bounds.) This difficulty can be addressed by ϵ-scaling, that is, by applying the algorithm several times, each time decreasing ϵ by a constant factor, up to the threshold value of $1/(N+1)$, while using the final prices obtained for one value of ϵ as starting prices for the next value of ϵ. A polynomial complexity bound of $O\big(N^2 A \log(NC)\big)$, where C is the cost range

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

can be proved for the resulting method. The unscaled version of the method, where ϵ is kept fixed at $1/(N+1)$, is pseudopolynomial. These complexity bounds can be derived using well-known lines of analysis [Ber86b], [BeE87], [Gol87], [BeE88], [BeT89], [GoT90], and will not be proved here.

3. VARIATIONS AND EFFICIENT IMPLEMENTATION

In this section we describe a number of variations of the algorithms of the preceding section, which we have empirically found to improve performance. Some of these variations are similar to corresponding variations of a related max-flow algorithm [Ber93].

Saving the Best Candidate

A number of implementation ideas have been shown to greatly accelerate the termination of the auction/shortest path algorithm [Ber91a], [Ber91b]. Some of these ideas are directly transferable to the minimum cost flow context, and are potentially very useful. In particular, the main computational bottleneck of the algorithm is the calculation of the best candidate arc for extension in Eq. (10), which is done every time node i becomes the terminal node of the path. We can reduce the number of these calculations by using the ϵ-CS condition

$$p_i \le \min\left\{ \min_{\{(i,j)\in\mathcal{A}\mid x_{ij}<c_{ij}\}} \{a_{ij}+p_j+\epsilon\},\ \min_{\{(j,i)\in\mathcal{A}\mid b_{ji}<x_{ji}\}} \{p_j-a_{ji}+\epsilon\} \right\}, \tag{13}$$

and the following observation: if some arc (i,j_i) satisfies

$$p_i = a_{ij_i}+p_{j_i}+\epsilon \qquad \text{and} \qquad x_{ij_i}<c_{ij_i} \tag{14}$$

it follows that

$$p_i = \min\left\{ \min_{\{(i,j)\in\mathcal{A}\mid x_{ij}<c_{ij}\}} \{a_{ij}+p_j+\epsilon\},\ \min_{\{(j,i)\in\mathcal{A}\mid b_{ji}<x_{ji}\}} \{p_j-a_{ji}+\epsilon\} \right\},$$

so if i is the terminal node, the path can be extended by j_i. The same is true if some arc (j_i,i) satisfies

$$p_i = a_{j_i i}-p_{j_i}+\epsilon \qquad \text{and} \qquad b_{j_i i}<x_{j_i i}. \tag{15}$$

This suggests the following implementation strategy: each time the minimum in Eq. (13) is calculated, we store an arc (i,j_i) such that Eq. (14) holds or an arc (j_i,i) such that Eq. (15) holds. At the next time node i becomes the terminal node of the path, we check whether Eq. (14) or (15), respectively, is still satisfied, and if it is, we extend the path by node j_i without going through the calculation of the minimum in Eq. (13). In practice this device is very effective.

Using a Second Best Candidate

Suppose that each time the minimum in Eq. (13) is calculated, we store an arc (i,j_i) such that Eq. (14) holds or an arc (j_i,i) such that Eq. (15) holds. Assume further that for the terminal node i of the current path P we have available a lower bound β_i on the value of the minimum in Eq. (13) over nodes j other than $j=j_i$, that is,

$$\min\left\{ \min_{\{(i,j)\in\mathcal{A}\mid x_{ij}<c_{ij},\, j\neq j_i\}} \{a_{ij}+p_j+\epsilon\},\ \min_{\{(j,i)\in\mathcal{A}\mid b_{ji}<x_{ji},\, j\neq j_i\}} \{p_j-a_{ji}+\epsilon\} \right\} \ge \beta_i. \tag{16}$$

Suppose also that the test for an extension is failed, that is, Eq. (13) holds with strict inequality. Then if the current "best" arc is (i,j_i), we can check to see whether we have

$$a_{ij_i}+p_{j_i}+\epsilon \le \beta_i \qquad \text{and} \qquad x_{ij_i}<c_{ij_i} \tag{17}$$

and if this is so we know that (i,j_i) is still the best arc, thus making the computation of the minimum of Eq. (13) unnecessary. An analogous statement holds if the current "best" arc is (j_i,i) and we have

$$a_{j_i i}-p_{j_i}+\epsilon \le \beta_i \qquad \text{and} \qquad b_{j_i i}<x_{j_i i}. \tag{18}$$

A lower bound β_i can be obtained by calculating, together with the "best" arc, a "second best" arc (i, j_i') [or (j_i', i)] in the minimization of Eq. (13), and a corresponding value $\beta_i = a_{ij_i'} + p_{j_i'} + \epsilon$ (or $\beta_i = a_{j_i'i} - p_{j_i'} + \epsilon$, respectively) out of those entering the minimization in Eq. (13). Then, because node prices are monotonically nondecreasing throughout the algorithm, as long as no new arc incident to i becomes admissible, we can use β_i as a suitable lower bound (if a new arc incident to i enters the adnmissible graph due to an augmentation, we must suitably modify β_i and the corresponding "second best" arc). Furthermore, if the test of Eqs. (17) and (18) is failed, we can check to see whether the second best arc (i, j_i') [or (j_i', i)] is still admissible and whether $a_{ij_i'} + p_{j_i'} + \epsilon = \beta_i$ (or $a_{j_i'i} - p_{j_i'} + \epsilon = \beta_i$, respectively). If this is so the "second best arc" becomes the "best" arc, thereby obviating again the calculation of the minimum in Eq. (13).

The idea of using a "second best" arc has been shown to be very effective in auction algorithms for the assignment problem ([Ber91a], p. 176), the shortest path problem [Ber91b], [CDP92], and max-flow problems [Ber93]. It similarly improves substantially the performance of the algorithm of this paper.

Saving Path Fragments

Suppose that following an augmentation that starts at a node s, a portion of P starting at s and ending at some node i is still unblocked, while the surplus of s is still positive. Then we can start the next iteration with the same node s, move directly to the terminal node i, and continue the search for an augmenting path from there. This variation, which was also discussed in a different context in [MPS91], saved a modest amount of computation time in our experiments.

Early Flow Augmentations

We have found empirically that the total number of price changes is reduced if the length of the current path (the number of arcs of the path) is not allowed to become too long. Under some circumstances, this can lead to the path P becoming alternatively short and long many times before an augmentation can occur. We have thus employed the heuristic of performing an augmentation along the current path, whenever a contraction occurs with an attendant label change of 2ϵ or less, and furthermore the number of arcs of the path is more than two.

Optimistic Extensions

In practice, it appears that the effectiveness of the algorithm is enhanced significantly if an extension is performed not just when

$$p_i = \min\left\{ \min_{\{(i,j)\in\mathcal{A} \mid x_{ij}<c_{ij}\}} \{a_{ij} + p_j + \epsilon\},\ \min_{\{(j,i)\in\mathcal{A} \mid b_{ji}<x_{ji}\}} \{p_j - a_{ji} + \epsilon\} \right\}, \tag{19}$$

but also when the weaker condition

$$p_i \geq \min\left\{ \min_{\{(i,j)\in\mathcal{A}\mid x_{ij}<c_{ij}\}} \{a_{ij}+p_j\},\ \min_{\{(j,i)\in\mathcal{A}\mid b_{ji}<x_{ji}\}} \{p_j - a_{ji}\} \right\} \tag{20}$$

holds. This maintains ϵ-CS, and allows the path to "extend faster" towards a negative surplus node, but introduces a difficulty: a cycle may be closed by extending the path, that is, the extension node j_i may already belong to P. One can bypass this difficulty by keeping track of which nodes belong to P and by checking for a potential cycle formation. Whenever a cycle is about to be closed by extension, a "retreat" operation is performed, which backtracks along the path and sets the price of each successive terminal node i to

$$\min\left\{ \min_{\{(i,j)\in\mathcal{A}\mid x_{ij}<c_{ij}\}} \{a_{ij}+p_j+\epsilon\},\ \min_{\{(j,i)\in\mathcal{A}\mid b_{ji}<x_{ji}\}} \{p_j - a_{ji}+\epsilon\} \right\} \tag{21}$$

up to the point where p_i strictly increases. Despite the overhead introduced by retreat operations, in our experiments, optimistic extensions resulted in considerable net saving in computation time.

A particularly interesting fact is that in the case of a max-flow problem, the retreat operations are unnecessary, that is, the path never closes a cycle even if the weaker criterion (20) is used for an extension. This is shown in [Ber93], where the corresponding path construction algorithm is studied in more detail and is embedded within the Ford-Fulkerson augmentation approach. A corresponding code, called AUCTION/MF and described in [Ber93], has compared very favorably with state-of-the-art FORTRAN implementations of the highest distance version of the preflow-push algorithm, written by Ahuja and Orlin, and Derigs and Meier [DeM89].

4. COMPUTATIONAL RESULTS

In this section we discuss some of our computational experience with three FORTRAN codes that use the algorithm of this paper. These are:

(a) AUCTION-(1-Sided): This is the algorithm of Section 2 and includes some of the variations discussed in Section 3, including ϵ-scaling and optimistic extensions.

(b) AUCTION-(2-Sided): This is the same code as AUCTION-(1-Sided), but it starts auction iterations not just from nodes with positive surplus but also from nodes with negative surplus. The latter iterations involve "reverse" augmentations, which "pull" flow from nodes of positive surplus towards nodes of negative surplus, and also involve contractions that decrease node prices. There have been similar proposals for two-sided ϵ-relaxation and auction algorithms in the literature, starting with the original paper on ϵ-relaxation [Ber86a].

(c) RELAX-AUCTION: This is the recently released version of the RELAX code, called RELAX-IV [BeT94], which uses (as an option) the code AUCTION-(1-Sided) to initialize the relaxation code of Bertsekas and Tseng [BeT88a], [BeT88b]. In particular, one or two scaling phases of the auction algorithm are performed with relative large value of ϵ, and the resulting prices are using as starting prices for the relaxation code after the flows are suitably modified to satisfy complementary slackness. We have used the default setting that specifies one scaling phase with an ϵ equal to 1/8 of the cost range of the problem. Extensive computational results with this code and comparisons with other codes have also been given in [BeT94].

We have compared these codes with the following FORTRAN codes on a variety of randomly generated test problems:

(1) RELAX: This is the RELAX-IV code described in [BeT94] without the auction initialization.

(2) NETFLO: This is the state-of-the-art primal simplex code due to Kennington and Helgason [KeH80].

(3) E-RELAX: This is the ϵ-relaxation code given in [Ber91a].

Summary of the Experimentation Results

While we have experimented with a broad variety of problems, much of the experimentation reported here involves problems for which the simplex method and, particularly, the relaxation method have significant difficulty and tend to be slow. These problems, exemplified by those of Tables 1 and 7, typically involve a graph of large diameter, such as a very sparse or a grid-like graph, and long augmenting paths. The difficulties of NETFLO and RELAX with such problems have been documented in [ReV93]. For "relatively easy" problems where the augmenting paths are relatively short (e.g., many types of transportation problems such as those of Table 5), RELAX performs extremely well and outperforms substantially the other codes discussed here. Our main conclusions are as follows:

(a) When the auction algorithm of this paper is used to initialize the relaxation method as in RELAX-AUCTION, the difficulties of the relaxation method with long augmenting paths are largely eliminated. As a result, RELAX-AUCTION is a very efficient and reliable code for both easy and difficult problems. As explained in [BeT94], the reason for this is that the first scaling phase of the auction algorithm moves quickly flow from the supply points of the network towards the demand points, and the resulting initial flow that is provided to the relaxation method involves relatively short path augmentations, which the relaxation method accomplishes very efficiently. This phenomenon was the main motivation for combining the auction and relaxation methods.

(b) The auction code AUCTION-(2-Sided) almost always outperforms E-RELAX, while AUCTION-(1-Sided) outperforms E-RELAX in the majority of problems. This is consistent with the findings of [Ber93] for max-flow problems, where the max-flow version of the algorithm of this paper outperforms efficient preflow-push methods by a large margin. Thus there is consistent evidence so far that auction algorithms based on multiple node path augmentations perform better than auction methods such as the ϵ-relaxation and preflow-push methods that involve single arc flow changes. It should be noted, however, that the ϵ-relaxation method is well-suited for parallelization [BeT89], [LiZ91], [NiZ93], while the parallelization potential of the methods of the present paper is somewhat unclear.

(c) For relatively difficult problems, the two-sided auction code often outperforms substantially the one-sided auction code. For relatively easy problems the two auction codes perform comparably.

(d) For relatively difficult problems, AUCTION-(2-Sided) outperforms in many cases all other codes, including RELAX-AUCTION, and sometimes by a substantial margin (see Tables 2 and 8). This is not true for all difficult problem types that we tested, and further experimentation with a broader variety of problems is required to reach reliable conclusions in this regard. However, our extremely favorable results on some difficult problems with the two-sided version of the auction algorithm of this paper was somewhat unexpected to us.

Experimentation Results

We present computational results with four types of problems. All experimentation reported here was done on a Sun Sparc 5 with 24 Megabytes of memory. We are thankful to Lakis Polymenakos who adapted the codes for the Sun Sparc 5 and conducted the experiments.

(1) Relatively sparse transhipment problems generated with NETGEN, the well-known and widely used problem generator of [KNS74]. Ten of these problems are described in Table 1, and the corresponding computation times are given in Table 2.

(2) Substantially less sparse transhipment problems than the problems of Table 1, also generated with NETGEN. Five such problems are described in Table 3, and the corresponding computation times are given in Table 4.

(3) Symmetric transportation problems generated with NETGEN. Five such problems are described in Table 5, and the corresponding computation times are given in Table 6.

(4) Grid-like graphs generated by the random problem generator GRIDGEN given in [Ber91a]. This generator provides a grid of arcs with wraparound, plus some additional arcs with randomly generated end nodes (the number of these latter arcs was $2N$ in our test problems). The randomly generated arcs have cost and capacity selected according to a uniform distribution from given ranges. The grid arcs have cost equal to the maximum cost of the range provided, and capacity equal to the total supply. In our experiments a single source node and a single node were randomly selected. Ten such problems are described in Table 7, and the corresponding computation times are given in Table 8.

REFERENCES

[BPS92] Bertsekas, D. P., Pallottino, S., and Scutella," M. G., "Polynomial Auction Algorithms for Shortest Paths," LIDS-P-2107, M.I.T., May 1992, Computational Optimization and Applications, to appear.

[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," Math. Programming, Series B, Vol. 42, 1988, pp. 203-243.

[BeT88a] Bertsekas, D. P., and Tseng, P., "Relaxation Methods for Minimum Cost Ordinary and Generalized Network Flow Problems," Operations Research, Vol. 36, 1988, pp. 93-114.

[BeT88b] 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.

[BeT94] Bertsekas, D. P., and Tseng, P., " RELAX-IV: A Faster Version of the RELAX Code for

Prob.	Nodes	Arcs	Cost Range	Total Surplus	Capacity Range
1	4000	12000	[1,1000]	10000	[200,2000]
2	5000	13000	[1,100]	100000	[1000,3000]
3	5000	15000	[1,10]	100000	[2000,5000]
4	10000	23000	[1,10]	20000	[200,2000]
5	11000	17000	[1,10]	100000	[2000,5000]
6	15000	32000	[1,10]	100000	[2000,5000]
7	20000	37000	[1,20]	200000	[4000,10000]
8	20000	37000	[1,20]	200000	[4000,12000]
9	25000	50000	[1,10]	100000	[2000,5000]
10	30000	80000	[1,100]	100000	[1000,5000]

Table 1: Sparse transhipment test problems generated by NETGEN.

Prob.	Nodes	Arcs	Cost Range	Total Surplus	Capacity Range
1	1000	15000	[1,1000]	100000	[2000,5000]
2	2000	35000	[1,1000]	100000	[1000,5000]
3	3000	60000	[1,1000]	100000	[1000,4000]
4	2000	40000	[1,1000]	100000	[2000,4000]
5	5000	75000	[1,100]	10000	[100,400]

Table 3: Relatively dense transhipment test problems generated by NETGEN.

Prob.	RELAX-AUCTION	RELAX	AUCTION 1-sided	AUCTION 2-sided	NETFLO	E-RELAX
1	2.43	27.05	2.15	1.16	1.61	2.93
2	3.89	25.24	31.75	5.9	8.02	24.39
3	2.86	3.75	6.18	2.24	3.68	8.59
4	3.95	4.45	11.83	2.79	8.09	13.51
5	9.04	9.45	67.95	8.73	11.04	44.6
6	7.3	8.7	35.51	5.09	11.32	22.37
7	8.6	10.44	41.61	8.2	18.09	36.00
8	8.46	8.11	40.41	8.32	16.98	31.3
9	15.66	14.42	62.78	8.19	19.81	50.5
10	15.386	15.5	46.02	7.14	34.00	27.88

Table 2: Computation times for sparse transhipment problems generated by NETGEN (cf. Table 1).

Prob.	RELAX-AUCTION	RELAX	AUCTION 1-sided	AUCTION 2-sided	NETFLO	E-RELAX
1	0.84	1.25	1.69	1.8	1.07	2.75
2	1.74	37.04	3.24	3.6	2.48	6.51
3	3.64	2.9	6.2	6.3	4.58	14.79
4	2.92	2.97	4.38	4.38	4.29	8.95
5	3.35	4.48	8.17	6.44	10.5	16.44

Table 4: Computation times for relatively dense transhipment problems generated by NETGEN (cf. Table 3).

Prob.	Nodes	Arcs	Cost Range	Total Surplus	Capacity Range
1	10000	20000	[1,100]	100000	[10,30]
2	20000	40000	[1,100]	100000	[30,70]
3	30000	60000	[1,100]	100000	[20,50]
4	40000	80000	[1,100]	200000	[10,70]
5	50000	100000	[1,100]	250000	[10,70]

Table 5: Symmetric transportation problems generated by NETGEN.

Prob.	RELAX-AUCTION	RELAX	AUCTION 1-sided	AUCTION 2-sided	NETFLO	E-RELAX
1	4.93	4.05	20.33	26.0	94.23	32.34
2	19.23	14.1	119.44	78.04	573.76	160.3
3	26.23	21.85	611.28	120.4	1401.64	815.55
4	39.59	33.24	263.32	192.3	3006.91	354.2
5	65.44	55.41	339.06	234.25	4839.33	482.47

Table 6: Computation times for symmetric transportation problems generated by NETGEN (cf. Table 5).

Solving Minimum Cost Flow Problems," Report LIDS-P-2276, Lab. for Information and Decision Systems, M.I.T., Nov. 1994.

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

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

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

Prob.	Nodes	Arcs	Cost Range	Total Surplus	Capacity Range
1	50×50	15000	[1,100]	10000	[1000,2000]
2	70×70	29400	[1,100]	10000	[1000,4000]
3	20×300	36000	[1,100]	10000	[2000,4000]
4	100×100	60000	[1,100]	10000	[1000,4000]
5	20×500	60000	[1,100]	10000	[2000,4000]
6	20×500	60000	[1,100]	10000	[1000,3000]
7	50×200	60000	[1,100]	10000	[1000,3000]
8	110×110	72600	[1,100]	10000	[2000,4000]
9	150×150	135000	[1,100]	10000	[2000,4000]
10	200×200	240000	[1,100]	10000	[2000,4000]

Table 7: Grid-like problems generated by GRIDGEN.

1986.

[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.

[Ber93] Bertsekas, D. P., "An Auction Algorithm for the Max-Flow Problem," Report LIDS-P-2193, Lab. for Information and Decision Systems, M.I.T., Aug. 1993 (J.O.T.A. to appear).

[CDP92] Cerulli, R., De Leone, R., and Piacente, G., "A Modified Auction Algorithm for the Shortest Path Problem," University of Salerno Report, to appear in Math. Optimization and Software.

[DeM89] Derigs, U., and Meier, W., "Implementing Goldberg's Max-Flow Algorithm – A Computational Investigation," Zeitschrif fur Operations Research, Vol. 33, pp. 383-403.

[FoF57] Ford, L. R., Jr., and Fulkerson, D. R., "A Primal-Dual Algorithm for the Capacitated Hitchcock Problem," Naval Res. Logist. Quart., Vol. 4,1957, pp. 47-54.

Prob.	RELAX-AUCTION	RELAX	AUCTION 1-sided	AUCTION 2-sided	NETFLO	E-RELAX
1	0.42	1.1	0.97	0.57	0.78	1.24
2	0.74	4.23	1.85	0.89	1.68	2.31
3	1.03	4.64	2.1	1.07	2.74	2.42
4	1.74	8.53	3.44	1.65	5.84	4.18
5	1.41	1.01	3.25	1.52	4.9	4.02
6	1.51	3.29	3.47	1.64	4.06	4.40
7	2.57	1.98	3.56	1.71	9.1	4.43
8	2.99	2.39	4.29	2.06	7.86	5.42
9	4.21	19.7	8.84	3.88	16.44	5.62
10	8.02	50.85	18.1	7.07	38.08	10.66

Table 8: Computation times for grid-like problems generated by GRIDGEN (cf. Table 8).

[FoF62] Ford, L. R., Jr., and Fulkerson, D. R., Flows in Networks, Princeton Univ. Press, Princeton, N. J., 1962

[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., Cambridge, MA., 1987.

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

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

[MPS91] Mazzoni, G., Pallotino, S., and Scutella', M. G., "The Maximum Flow Problem: A Max-Preflow Approach," European J. of Operational Research, Vol. 53, 1991, pp. 257-278.

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

[PaS91] Pallottino, S., and Scutella', M. G., "Strongly Polynomial Algorithms for Shortest Paths," Ricerca Operativa, Vol. 60, 1991, pp. 33-53.

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

[ReV93] Resende, M. G. C., and Veiga, G., "An Efficient Implementation of a Network Interior Point Method," in Network Flow and Matching: First DIMACS Implementation Challenge, Edited by D. Johnson and C. McGeoch, American Mathematical Society, Providence, RI, 1993, pp. 299-348.

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