Mathematical Programming 42 (1988) 203-243
North-Holland

DUAL COORDINATE STEP METHODS FOR LINEAR NETWORK FLOW PROBLEMS

Dimitri P. BERTSEKAS and Jonathan ECKSTEIN

Laboratory for Information and Decision Systems and Operations Research Center, Massachusetts Institute of Technology, Cambridge, MA 02139, USA

Received 12 August 1987
Revised manuscript received 2 June 1988

We review a class of recently-proposed linear-cost network flow methods which are amenable to distributed implementation. All the methods in the class use the notion of ϵ-complementary slackness, and most do not explicitly manipulate any "global" objects such as paths, trees, or cuts. Interestingly, these methods have stimulated a large number of new *serial* computational complexity results. We develop the basic theory of these methods and present two specific methods, the *ϵ-relaxation* algorithm for the minimum-cost flow problem, and the *auction* algorithm for the assignment problem. We show how to implement these methods with serial complexities of $O(N^3 \log NC)$ and $O(NA \log NC)$, respectively. We also discuss practical implementation issues and computational experience to date. Finally, we show how to implement ϵ-relaxation in a completely asynchronous, "chaotic" environment in which some processors compute faster than others, some processors communicate faster than others, and there can be arbitrarily large communication delays.

Key words: Network flows, relaxation, distributed algorithms, complexity, asynchronous algorithms.

1. Introduction

This paper considers a number of recent developments in network optimization, all of which originated from efforts to construct parallel or distributed algorithms. One obvious idea is to have a processor (or virtual processor) assigned to each node of the problem network. The intricacies of coordinating such processors makes it awkward to manipulate the "global" objects—such as cuts, trees, and augmenting paths—that are found in most traditional network algorithms. As a consequence, algorithms designed for such distributed environments tend to use only *local* information: the dual variables associated with a node and its neighbors, and the flows on the arcs incident to the node. For reasons that will become apparent later, we call this class of methods *dual coordinate step methods*. Their appearance has also stimulated a flurry of advances in *serial* computational complexity results for network optimization problems.

Supported by Grant NSF-ECS-8217668 and by the Army Research Office under grant DAAL03-86-K-0171. Thanks are due to David Castañon, Paul Tseng, and Jim Orlin for their helpful comments.

Another feature of these algorithms is that they all use a notion called *ε-complementary slackness.* As we shall see, this idea is essential to making sure that a method that uses only local information does not "jam" or halt at a suboptimal point. However, ε-complementary slackness is also useful in the construction of scaling algorithms. The combination of scaling and ε-complementary slackness has given rise to a number of computational complexity results, most of them serial. Some of the algorithms behind these results use only local information, but others use global data, usually to construct augmenting paths.

Here, we will concentrate on local algorithms, since they are the ones which hold the most promise of efficient parallel implementation, and show how they can be regarded as approximate coordinate ascent or relaxation methods in an appropriately-formulated dual problem. Section 2 of this paper gives an overview and partial history of these methods. Section 3 examines in detail what is perhaps the generic algorithm of the class, the ε-relaxation method [7]. Section 4 develops some basic serial complexity analysis tools for this algorithm [28, 29, 8], also addressing the special case of maximum flow problems. Section 5 combines this analysis with the notion of scaling [30–32, 19, 24, 41, 5], yielding a polynomial ($O(N^3 \log NC)$) serial algorithm for the minimum-cost flow problem (N is the number of nodes, and C the largest absolute value of the arc cost coefficients). In Section 6, we introduce the *auction algorithm* [9] for the assignment problems, and show how it may be regarded as an implementation of a special form of ε-relaxation in which nodes are processed in a particular order. In view of this connection, we indicate how the auction algorithm can be implemented in $O(NA \log NC)$ time, where A is the number of arcs in the network. We present computational results indicating that the practical performance of the auction algorithm is competitive with state-of-the-art codes (unfortunately, the same cannot yet be said of ε-relaxation). In Section 7, we present an implementation of ε-relaxation that works in a completely asynchronous, chaotic environment [10]. Finally, Section 8 presents conclusions and discusses some of the open questions regarding this class of algorithms.

2. History and overview

We first introduce the minimum-cost flow problem and its dual. Consider a directed graph with node set N and arc set A, with each arc (i, j) having a cost coefficient a_{ij}. Letting f_{ij} be the flow of the arc (i, j), the classical min-cost flow problem [39, Ch. 7] may be written

$$\text{minimize} \sum_{(i,j)\in A} a_{ij} f_{ij} \tag{MCF}$$

subject to

$$\sum_{(i,j)\in A} f_{ij} - \sum_{(j,i)\in A} f_{ji} = s_i \quad \forall i \in N, \tag{1}$$

$$b_{ij} \leq f_{ij} \leq c_{ij} \quad \forall (i,j) \in A, \tag{2}$$

where a_{ij}, b_{ij}, c_{ij} and s_i, are given *integers*. In order for the constraints (1) to be consistent, we require that $\sum_{i\in N} s_i = 0$. We also assume that there exists at most one arc in each direction between any pair of nodes, but this assumption is for notational convenience and can be easily dispensed with. We denote the numbers of nodes and arcs by N and A, respectively. Also, let C denote the maximum absolute value of the cost coefficients, $\max_{(i,j)\in A}|a_{ij}|$.

In this paper, a *flow* f will be any vector in $\mathbb{R}^A$, with elements denoted f_{ij}, $(i,j)\in A$. A *capacity-feasible* flow is one obeying the capacity constraints (2). If a capacity-feasible flow also obeys the conservation constraints (1), it is a *feasible flow*.

We formulate a dual problem to (MCF) by associating a Lagrange multiplier p_i with each conservation of flow constraint (1). Letting f be a flow and p be the vector with elements p_i, $i\in N$, we can write the corresponding Lagrangian function as

$$L(f,p)=\sum_{(i,j)\in A}(a_{ij}+p_j-p_i)f_{ij}+\sum_{i\in N}s_ip_i. \tag{3}$$

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

$$\text{maximize } q(p) \tag{4}$$

subject to no constraint on p, with the dual functional q given by

$$\begin{aligned} q(p) &= \min_f\{L(f,p)\,|\,b_{ij}\le f_{ij}\le c_{ij},\ (i,j)\in A\} \\ &= \sum_{(i,j)\in A} q_{ij}(p_i-p_j)+\sum_{i\in N}s_ip_i \end{aligned} \tag{5a}$$

where

$$q_{ij}(p_i-p_j)=\min_{f_{ij}}\{(a_{ij}+p_j-p_i)f_{ij}\,|\,b_{ij}\le f_{ij}\le c_{ij}\}. \tag{5b}$$

The function q_{ij} is shown in Fig. 1. This formulation of the dual problem is consistent with conjugate duality frameworks [42], [43] but can also be obtained via linear programming duality theory [35], [39]. We henceforth refer to (MCF) as the *primal problem*, and note that standard duality results imply that the optimal primal cost equals the optimal dual cost. We refer to the dual variable p_i as the *price of node i*.

Naive coordinate ascent and the jamming phenomenon

We have now obtained a dual problem which is piecewise-linear and unconstrained. A straightforward approach to distributed unconstrained optimization is to have one processor responsible for maximization along each coordinate direction. This approach leads to an iterative algorithm, called *naive coordinate ascent*, that chooses at each iteration a node i, and maximizes the dual function q with respect to p_i, while keeping all other prices constant. Unfortunately, as shown in Fig. 2, this algorithm does not always work as desired, due to the nondifferentiability of q. In particular, there may be suboptimal price vectors p from which q cannot be improved by changing any *single* price coordinate. If naive coordinate ascent encounters such a point, it will loop infinitely without improving the dual objective; we call this phenomenon *jamming*.

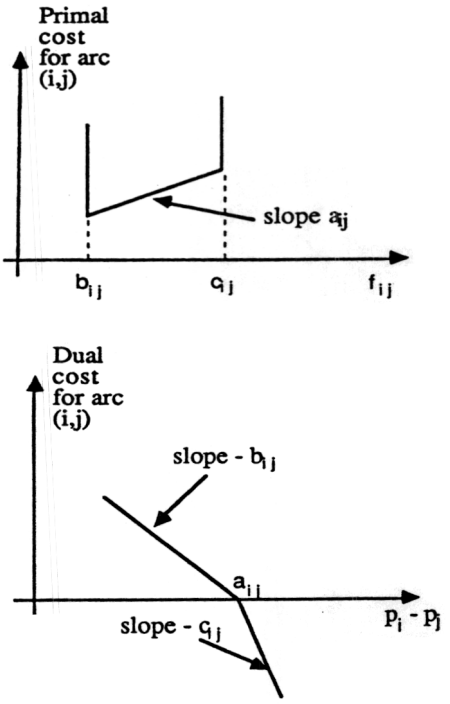

Fig. 1. Primal and dual costs for arc (i, j).

6

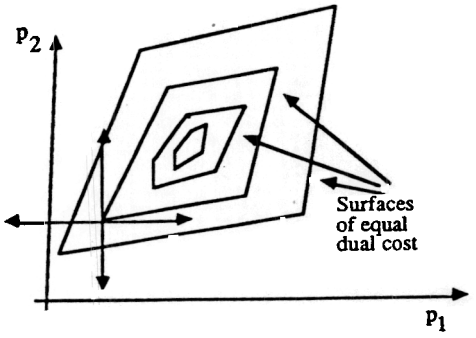

Fig. 2. Illustration of jamming. At the indicated point, it is impossible to improve the cost by changing any *single* price.

Jamming and the RELAX approach

One way of avoiding the jamming problem is embodied in the RELAX family of serial computer codes (see [12, 17, 46]). Essentially, these codes make dual ascents along directions that have a minimal number of non-zero components, which means that they select coordinate directions whenever possible. Only when jamming occurs

do they select more complicated ascent directions. These codes have proved remarkably efficient in practice; however, adapting them to exploit a massively parallel computing environment appears to be a very intricate task, due to the difficulties of coordinating many simultaneous multiple-node price change and labeling operations.

Note that jamming would not occur if the dual cost were differentiable. If the primal cost function is *strictly* convex, then the dual cost is indeed differentiable, and application of coordinate ascent is straightforward and well-suited to parallel implementation. Proposals for methods of this type include [44, 20, 38, 22 and 36]. [48] contains computational results on a simulated parallel architecture, and [47] results on an actual parallel machine. [14] and [15] contain convergence proofs.

The auction approach

A different, more radical approach to the jamming problem is to allow small price changes, say by some amount ε, even if they worsen the dual cost. This idea dates back to the 1979 *auction algorithm* [9, 10, 11], a procedure for the assignment (bipartite matching) problem that predates the RELAX family of algorithms. (An extension to the transportation problem is given in [13].) In this algorithm, one considers the nodes on one side of the bipartite graph to be "people" or agents placing bids for the "objects" representing the nodes on the other side of the graph. The dual variables p_j corresponding to the "object" nodes may then be considered to be the actual current prices of the objects in the auction. The phenomenon of jamming in this context manifests itself as two or more people submitting the same bid for an object. In a real auction, such conflicts are resolved by people submitting slightly higher bids, thus raising the price of the object, until all but one bidder drops out and the conflict is resolved (we give a more rigorous description of the auction algorithm later in this paper).

ε-relaxation and ε-complementary slackness

This idea of resolving jamming by forcing (small) price increases even if they worsen the dual cost is also fundamental to the central algorithm of this paper, which we call *ε-relaxation.* This algorithm was first introduced in [7]; [8] is a revision of [7] which includes a (non-polynomial) complexity analysis of the algorithm without the use of scaling.

To develop this algorithm, we introduce the classical complementary slackness conditions, and a relaxation of them which we call *ε-complementary slackness.* The classical complementary slackness conditions for minimum-cost network flow problem may be expressed as

$$f_{ij} < c_{ij} \Rightarrow p_i - p_j \leq a_{ij} \quad \forall (i,j) \in A, \tag{6a}$$

$$b_{ij} < f_{ij} \Rightarrow p_i - p_j \geq a_{ij} \quad \forall (i,j) \in A. \tag{6b}$$

Standard linear programming duality theory gives that f and p are jointly optimal for the primal and dual problems, respectively, if and only if they satisfy complementary slackness and f is feasible.

Appealing to conjugate duality theory [42, 43], there is a useful interpretation of the complementary slackness conditions (6a-b). Referring to Fig. 1, the complementary slackness conditions on (i, j) and the capacity constraint $b_{ij} \leq f_{ij} \leq c_{ij}$ are precisely equivalent to requiring that $-f_{ij}$ be a supergradient of the dual function component q_{ij} at the point $p_i - p_j$. This may be written $-f_{ij} \in \partial q_{ij}(p_i - p_j)$. Adding these conditions together for all arcs incident to a given node i and using the definition of the dual functional (5a), one obtains that for any pair (f, p) obeying complementary slackness, the *surplus* of node i, defined to be

$$g_i = \sum_{(j,i)\in A} f_{ji} - \sum_{(i,j)\in A} f_{ij} + s_i, \tag{7}$$

is in fact a supergradient of $q(p)$ considered as a function of p_i, with all other node prices held constant. We may express this as $g_i \in \partial q_i(p_i; p)$, where $q_i(\cdot; p)$ denotes the function of a single variable obtained from q by holding all prices except the ith fixed at p. The surplus also has the interpretation as the flow into node i minus the flow out of i given by the (possibly infeasible) flow f. Thus a flow f is feasible if and only if the corresponding surpluses g_i are zero for all $i \in N$. (Note that the *sum* of all the surpluses is zero for *any* flow.)

We make a few further definitions: we say that an arc (i, j) is

$$\textit{Inactive} \quad \text{if } p_i < a_{ij} + p_j, \tag{8a}$$

$$\textit{Balanced} \quad \text{if } p_i = a_{ij} + p_j, \tag{8b}$$

$$\textit{Active} \quad \text{if } p_i > a_{ij} + p_j. \tag{8c}$$

The combined condition $-f_{ij} \in \partial q_{ij}(p_i - p_j)$ may then be reexpressed as

$$f_{ij} = b_{ij} \quad \text{if } (i,j) \text{ active}, \tag{9a}$$

$$b_{ij} \leq f_{ij} \leq c_{ij} \quad \text{if } (i,j) \text{ balanced}, \tag{9b}$$

$$f_{ij} = c_{ij} \quad \text{if } (i,j) \text{ active}. \tag{9c}$$

Figure 3 displays the form of the dual function along a single price coordinate p_i.

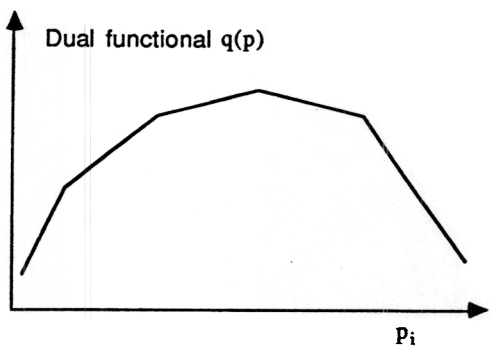

Fig. 3. The dual functional $q(p)$ graphed with respect to a single price coordinate.

The breakpoints along the curve correspond to points where one or more arcs incident to node i are balanced. If one wishes to maintain complementary slackness, only at the breakpoints is there any freedom in choosing arc flows; on the linear portions of the graph, all arcs are either active or inactive, and all flows are determined exactly by (9a) and (9c).

Now consider the classical complementary slackness conditions relaxed by a nonnegative amount ε, thus:

$$f_{ij} < c_{ij} \Rightarrow p_i - p_j \leq a_{ij} + \varepsilon, \tag{10a}$$

$$b_{ij} < f_{ij} \Rightarrow p_i - p_j \geq a_{ij} - \varepsilon. \tag{10b}$$

A flow-price pair (f, p) obeying these relaxed conditions is said to obey *ε-complementary slackness.* The notion of ε-complementary slackness was used in [9, 10], and introduced more formally in [15, 17]. It was also used in the analysis of [45] (Lemma 2.2) in the special case where the flow vector f is feasible. Figure 4 compares the "kilter diagrams" for conventional and ε-complementary slackness.

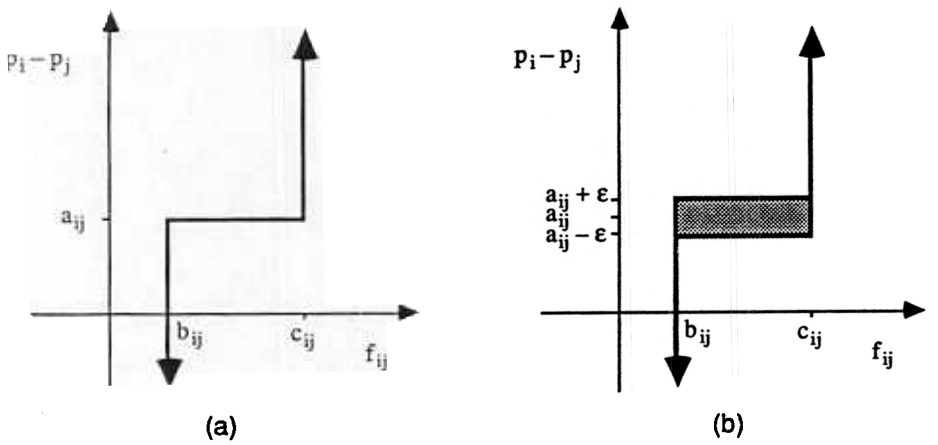

Fig. 4. Kilter diagrams for (a) conventional complementary slackness and (b) ε-complementary slackness.

The ε-relaxation algorithm works by maintaining a flow-price pair (f, p), with f integral, that obeys ε-complementary slackness, but not necessarily regular complementary slackness. It repeatedly selects nodes i whose surplus g_i is *positive,* and sets the corresponding price p_i to a value which is within ε of some maximizer of the dual cost with respect to p_i, with all other prices held constant (usually this value is ε plus the largest maximizer of the dual cost if such a maximizer exists). This operation is called an *up iteration.* Because p_i is not set to the maximizing value, the method does not necessarily obtain an improvement in the dual objective at each iteration; thus we call it a dual coordinate *step* method as opposed to a coordinate *ascent* method. With each up iteration, the flow vector is adjusted to maintain integrality and ε-complementary slackness. As we shall prove below, this process will eventually drive all the nodes' surpluses to zero, resulting in a final

flow f that is optimal if $\varepsilon < d/N$, where d is the greatest common divisor of the arc costs. It avoids jamming by following paths such as those depicted in Fig. 5. If $\varepsilon \geq d/N$, the algorithm will still terminate with a feasible flow, but this flow may not be optimal.

The Goldberg-Tarjan maximum flow method

Another important algorithm belonging to the dual coordinate step class is the maximum flow method of Goldberg and Tarjan [28, 29]. This algorithm was developed roughly concurrently with, and entirely independently from, the auction algorithm and the RELAX family of codes. The original motivation for this algorithm seems to have been quite different from the theory we emphasize in this paper; it appears to have been originally conceived of as a distributed, approximate computation of the "layered" representation of the residual network that is common in maximum flow algorithms [24]. However, it turns out that the first phase of this two-phase algorithm, in its simpler implementations, is virtually identical to ε-relaxation as applied to a specific formulation of the maximum flow problem. This connection will become apparent later. Basically, the distance estimates of the maximum flow algorithm may be interpreted as dual variables, and the method in fact maintains ε-complementary slackness with $\varepsilon = 1$.

The connection between the Goldberg-Tarjan maximum flow and ε-relaxation provides two major benefits: ε-relaxation gives a natural, straightforward way of reducing the maximum flow method to a single phase, and much of the maximum flow method's complexity analysis can be applied to the case of ε-relaxation.

Complexity analysis

There are several difficulties in adapting the maximum flow analysis of [28] and [29] to the case of ε-relaxation. The first is in placing a limit on the amount that prices can rise. The approach taken here synthesizes the ideas of [8] with those of

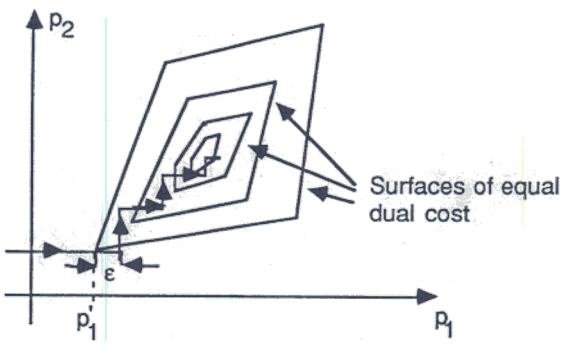

Fig. 5. When the ith price p_i is chosen for relaxation, it is changed to $p_i' + \varepsilon$, where p_i' is a value of the ith price that maximizes $q(p)$ with all other prices held fixed. When ε is small, it is possible to approach the optimal solution even if each step does not result in a dual cost improvement. The method eventually stays in a small neighborhood of the optimal solution.

[30]-[32]. This methodology can also be applied directly to solving maximum flow problems with arbitrary initial prices, as opposed to initial prices satisfying $p_i \leq p_j + 1$ for all arcs (i, j), as in [28] and [29].

Another problem is that of *flow looping*. This is discussed in Section 4 (see Fig. 7), and refers to a phenomenon whereby small increments of flow move an exponential number of times around a loop without any intermediate price changes. To overcome this difficulty one must initialize the algorithm in a way that the subgraph of arcs along which flow can change is acyclic at all times. In the max-flow problem this subgraph is naturally acyclic, so the difficulty does not arise. Flow looping is also absent from the assignment problem because all arcs may be given a capacity of 1, and (as we shall see) the algorithm changes flows only by integer amounts.

Section 4 also discusses the problem of *relaxing nodes out of order*. The acyclic subgraph mentioned above defines a partial order among nodes, and it is helpful to operate on nodes according to this order. This idea is central in the complexity analysis of [8], and leads to a simple and practical implementation that maintains the partial order in a linked list. We call this the *sweep implementation*. This analysis, essentially given in [8], provides an $O(N^2\beta/\varepsilon)$ complexity bound where β is a parameter bounded by the maximum simple path length in the network where the length of arc (i, j) is taken to be $|a_{ij}|$. Maximum flow problems can be formulated so that $\beta/\varepsilon = O(N)$, giving an $O(N^3)$ complexity bound for essentially arbitrary initial prices. For other minimum cost flow problems, including the assignment problem, the complexity is pseudopolynomial, being sensitive to the arc cost coefficients. The difficulty is due to a phenomenon which we call *price haggling*. This is analogous to the ill-conditioning phenomenon in unconstrained optimization, and is characterized by an interaction in which several nodes restrict one another from making large price changes (see Sections 5 and 6).

Developments in scaling

ε-complementary slackness is also useful in constructing scaling algorithms, which conversely help to overcome the problem of price haggling. We first distinguish between two kinds of scaling: *cost scaling* and *ε-scaling*. In cost scaling algorithms (which have their roots in [24]), one holds ε fixed and gradually introduces more and more accurate cost data; in ε-scaling, the cost data are held fixed and ε is gradually reduced. In both cases, the solutions obtained at the end of each scaling phase (except the last) may not be optimal for the cost data used for that phase, because ε may be greater than or equal to d/N. ε-scaling is mentioned in [9] as a method for improving the performance of the auction algorithm, based on computational experimentation. The method of ε-scaling was first *analyzed* in [30], where an algorithm with $O(NA \log(N) \log(NC))$ complexity was proposed, and a contrast with the method of cost scaling was drawn. The complexity of this algorithm was fully established in [31] and [32], where algorithms with $O(N^{5/3}A^{2/3} \log(NC))$ and $O(N^3 \log NC)$ complexity were also given, and parallel versions were also discussed.

The first two algorithms use complex, sophisticated data structures, while the $O(N^3 \log(NC))$ algorithm makes use of the sweep implementation. Both also employ a variation of ε-relaxation we call *broadbanding*, which will be described later in this paper. Independent discovery of the sweep implementation, following the appearance of [8], is claimed in [32] (where it is called the *wave implementation*). These results improved on the complexity bounds of all alternative algorithms for (MCF), which in addition are not as well suited for parallel implementation as the ε-relaxation method. Scaling analyses similar to [30] appeared later in such works as [26], [27], and [2].

In this paper we show how to moderate the effect of price haggling by using a similar but more traditional cost scaling approach in place of ε-scaling. This approach, given in [5], in conjunction with the sweep implementation, leads to a simple algorithm with an $O(N^3 \log(NC))$ complexity. It also bypasses the need for the broadbanding modification to the basic form of the ε-relaxation method, introduced in [30–32] in conjunction with ε-scaling.

Usually the most challenging part of scaling analysis [19, 24, 30–32, 37, 41, 45] is to show how the solution of one subproblem can be used to obtain the solution of the next subproblem relatively quickly. Here, the main fact is that the final price-flow pair (p, f) of one subproblem violates the ε-CS conditions for the next one by only a small amount. A way of taking advantage of this was first proposed in Lemmas 2–5 of [30] (see also [31, 32]). A key lemma is Lemma 5 of [30], which shows that the number of price changes per node needed to obtain a solution of the next subproblem is $O(N)$. There is a similar lemma in [19] that bounds the number of maximum flow computations in a scaling step in an $O(N^4 \log C)$ algorithm based on the primal-dual method. Our Lemma 5 of this paper is a refinement of Lemma 5 of [30], and is also an extension of Corollary 3.1 of [8], which bounds the number of price rises per node in the case where scaling is not used. We introduce a measure $\beta(p^0)$ of suboptimality of the initial price vector p^0, whereas [30–32] use an upper bound on this measure. This extension allows the lemma to be used in contexts other than scaling.

Other recent developments in scaling include Gabow and Tarjan's [27], which is also a cost scaling method. Furthermore, analysis in [32], drawing on some ideas of Tardos [45], shows that a *strongly polynomial* bound (that is, one polynomial in N and A) may be placed on a properly implemented scaling algorithm.

3. The ε-relaxation method in detail

To discuss ε-relaxation in detail, we must further develop the theory of ε-complementary slackness. We introduce some further terminology that will be useful later. We say that the arc (i, j) is

$$\varepsilon\text{-Inactive} \qquad \text{if } p_i < a_{ij} + p_j - \varepsilon, \tag{11a}$$

ε^--Balanced if $p_i = a_{ij} + p_j - \varepsilon$, (11b)

ε-Balanced if $a_{ij} + p_j - \varepsilon \leq p_i \leq a_{ij} + p_j + \varepsilon$, (11c)

ε^+-Balanced if $p_i = a_{ij} + p_j + \varepsilon$, (11d)

ε-Active if $p_i > a_{ij} + p_j + \varepsilon$. (11e)

Note that ε^--balanced and ε^+-balanced are both special cases of ε-balanced. The ε-complementary slackness conditions (combined with the arc capacity conditions) may now also be expressed as

$f_{ij} = b_{ij}$ if (i,j) is ε-inactive, (12a)

$b_{ij} \leq f_{ij} \leq c_{ij}$ if (i,j) is ε-balanced, (12b)

$f_{ij} = c_{ij}$ if (i,j) is ε-active. (12c)

The usefulness of ε-complementary slackness is evident in the following proposition:

Proposition 1. *If $\varepsilon < 1/N$, f is primal feasible (it meets both constraints* (1) *and* (2)*), and f and p jointly satisfy* ε-CS, *then f is optimal for* (MCF).

Proof. If f is not optimal then there must exist a simple directed cycle along which flow can be increased while the primal cost is improved. Let Y^+ and Y^- denote the sets of arcs of forward and backward arcs in the cycle, respectively. Then we must have

$$\sum_{(i,j)\in Y^+} a_{ij} - \sum_{(i,j)\in Y^-} a_{ij} < 0,$$

$$f_{ij} < c_{ij} \quad \text{for } (i,j) \in Y^+,$$

$$b_{ij} < f_{ij} \quad \text{for } (i,j) \in Y^-.$$

Using (10a-b), we have

$p_i \leq p_j + a_{ij} + \varepsilon$ for $(i,j) \in Y^+$, (14a)

$p_j \leq p_i - a_{ij} + \varepsilon$ for $(i,j) \in Y^-$. (14b)

Adding all the inequalities (14a) and (14b) together and using the hypothesis $\varepsilon < 1/N$ yields

$$\sum_{(i,j)\in Y^+} a_{ij} - \sum_{(i,j)\in Y^-} a_{ij} \geq -N\varepsilon > -1.$$

Since the a_{ij} are integral, this contradicts (13a). □

A strengthened form of Proposition 1 also holds when the arc cost coefficients and flow bounds are not integer, and is obtained by replacing the condition $\varepsilon < 1/N$ with the condition

$$\varepsilon < \min_{\text{All directed cycles } Y} \left\{ \frac{-\text{Length of } Y}{\text{Number of arcs in } Y} \quad \text{Length of } Y < 0 \right\},$$

where

$$\text{Length of cycle } Y = \sum_{(i,j)\in Y^+} a_{ij} - \sum_{(i,j)\in Y^-} a_{ij}. \tag{16}$$

The proof is obtained by suitably modifying the last relation in the proof of Proposition 1. A very useful special case is that f is optimal if $\varepsilon < d/N$, where d is the greatest common divisor of all the arc costs. When all arc costs are integer, we are assured that $d \geq 1$.

A useful way to think about ε-complementary slackness is that if the pair (f, p) obey it, then the rate of decrease in the primal cost to be obtained by moving flow around a directed cycle Y without violating the capacity constraints is at most $|Y|\varepsilon$. It limits the steepness of descent along the *elementary directions* (using the terminology of [42]) of the primal space.

The admissible graph

When the ε-relaxation algorithm is performing an iteration at some node i, it can only change the flow on two kinds of arcs: outgoing ε^+-balanced arcs (i, j) with $f_{ij} < c_{ij}$, and incoming ε^--balanced arcs (j, i) with $f_{ji} > b_{ji}$. We call these two kinds of arcs *admissible.* The *admissible graph* G^* corresponding to a pair (f, p) is the directed graph with node set N, an arc (i, j) for each ε^+-balanced arc (i, j) in A with $f_{ij} < c_{ij}$, and a reverse arc (j, i) for each ε^--balanced arc (i, j) in A with $f_{ij} > b_{ij}$. It is similar to the residual graph corresponding to the flow f which has been used by many other authors (see [39], for example), but only contains arcs that are admissible. Note that it is possible for the admissible graph to contain two distinct copies of the arc (i, j), one corresponding to (i, j) in the original network, and the other to (j, i).

Push lists

To obtain an efficient implementation of ε-relaxation, one must store a representation of the admissible graph. We use a simple "forward star" scheme in which each node i stores a linked list containing all the arcs of the original network corresponding to arcs of the admissible graph outgoing from i—that is, all arcs whose flow can be changed by iterations at i without any alteration in p. We call this list a *push list.* Although it is possible to maintain all push lists exactly at all times, doing so requires manipulating unnecessary pointers; it is more efficient to allow some inadmissible arcs to creep onto the push lists. However, all push lists must be *complete*: that is, though it may contain some extra arcs, i's push list must contain *every* arc whose flow can be altered by iterations at i without a price change.

The complexity results in most of the earlier work on the dual coordinate step class of algorithms [28, 29, 8, 30] implicitly require push lists or something similar. The first time push lists seem to have been discussed explicitly is in [32], where they are called *edge lists.*

The Exact form of up iteration

We now give a precise implementation of the up iteration. Assume that f is a capacity-feasible flow, the pair (f, p) obeys ε-complementary slackness, a push list corresponding to (f, p) exists at each node, and all these lists are complete. Let $i \in N$ be a node with positive surplus ($g_i > 0$).

Up iteration

Step 1 (Find Admissible Arc): Remove arcs from the top of i's push list until finding one which is still admissible (this arc is *not* deleted from the list). If $g_i > 0$ and the arc so found is an outgoing arc (i, j), go to Step 2. If $g_i > 0$ and the arc found is an incoming arc (j, i), go to Step 3. If the push list has become empty, go to Step 4. If an arc was found but $g_i = 0$, stop.

Step 2 (Decrease surplus by increasing f_{ij}): Set

$$f_{ij} := f_{ij} + \delta,$$

$$g_i := g_i - \delta,$$

$$g_j := g_j + \delta,$$

where $\delta = \min\{g_i, c_{ij} - f_{ij}\}$. If $\delta = c_{ij} - f_{ij}$, delete (i, j) from i's push list (it must be the top item). Go to Step 1.

Step 3 (Decrease surplus by reducing f_{ji}): Set

$$f_{ji} := f_{ji} - \delta,$$

$$g_i := g_i - \delta,$$

$$g_j := g_j + \delta,$$

where $\delta = \min\{g_i, f_{ji} - b_{ji}\}$. If $\delta = f_{ji} - b_{ji}$, delete (j, i) from i's push list (it must be the top item). Go to Step 1.

Step 4 (Scan/Price Increase): By scanning all arcs incident to i, set

$$p_i := \min\{\{p_j + a_{ij} + \varepsilon \,|\, (i, j) \in A \text{ and } f_{ij} < c_{ij}\} \cup \{p_j - a_{ji} + \varepsilon \,|\, (j, i) \in A \text{ and } b_{ji} < f_{ji}\}\} \tag{17}$$

and construct a new push list for i, containing exactly those incident arcs which are admissible with the new value of p_i. Go to Step 1. (*Note*: If the set over which the minimum in (17) is taken is empty and $g_i > 0$, halt with the conclusion that the problem is infeasible—see the comments below. If this set is empty and $g_i = 0$, increase p_i by ε and stop.)

The serial ε-relaxation algorithm consists of repeatedly selecting nodes i with $g_i > 0$, and performing up iterations at them. The method terminates when $g_i \leq 0$ for all $i \in N$, in which case it follows that $g_i = 0$ for all $i \in N$, and that f is feasible.

Basic lemmas

To see that execution of Step 4 must lead to a price *increase* note that when it is entered, we have

$$f_{ij} = c_{ij} \quad \text{for all } (i,j) \text{ such that } p_i \geq p_j + a_{ij} + \varepsilon, \tag{18a}$$

$$b_{ji} = f_{ji} \quad \text{for all } (j,i) \text{ such that } p_i \geq p_j - a_{ji} + \varepsilon. \tag{18b}$$

Therefore, when Step 4 is entered we have

$$p_i < \min\{p_j + a_{ij} + \varepsilon \,|\, (i,j) \in A \text{ and } f_{ij} < c_{ij}\}, \tag{19a}$$

$$p_i < \min\{p_j - a_{ji} + \varepsilon \,|\, (j,i) \in A \text{ and } b_{ji} < f_{ji}\}. \tag{19b}$$

It follows that Step 4 must increase p_i, unless $g_i > 0$ and the set over which the minimum is taken is empty. In that case, $f_{ij} = c_{ij}$ for all (i,j) outgoing from i and $b_{ji} = f_{ji}$ for all (j,i) incoming to i, so the maximum possible flow is going out of i while the minimum possible is coming in. If $g_i > 0$ under these circumstances, then the problem instance must be infeasible.

Lemma 1. *The ε-relaxation algorithm preserves the integrality of f, the ε-complementary slackness conditions, and the completeness of all push lists at all times. All node prices are monotonically nondecreasing throughout the algorithm.*

Proof. By induction on the number of up iterations. Assume that all the conditions hold at the outset of an iteration at node i. From the form of the up iteration, all changes to f are by integer amounts and ε-complementary slackness is preserved. By the above discussion, the iteration can only raise the price of i. Only inadmissible arcs are removed from i's push list in Steps 1, 2 and 3, and none of these steps change any prices; therefore, Steps 1, 2, and 3 preserve the completeness of push lists. In Step 4, i's push list is constructed exactly, so that push list remains complete. Finally, we must show that the price rise at i does not create any new admissible arcs that should be on *other* nodes' push lists. First, suppose $(j,i) \in A$ becomes ε^+-balanced as a result of a price rise at i. Then (j,i) must have been formerly ε-active, hence $f_{ji} = c_{ji}$, and (j,i) cannot be admissible. A similar argument applies to any (i,j) that becomes ε^--balanced as a result of a price rise at i. We have thus shown that all arcs added to the admissible graph by Step 4 are outgoing from i. □

Lemma 2. *Suppose that the initial prices p_i and the arc cost coefficients a_{ij} are all integer multiples of ε. Then every execution of Step 4 results in a price rise of at least ε, and all prices remain multiples of ε throughout the ε-relaxation algorithm.*

Proof. It is clear from the form of the up iteration that it preserves the divisibility of all prices by ε. Thus any price increase must be by at least ε, and the above discussion assures that every execution of Step 4 results in a price increase. The lemma follows by induction on the number of up iterations. □

We henceforth assume that all arc costs and initial prices are integer multiples of ε. A straightforward way to do this, considering the standing assumption that the a_{ij} are integer, is to let $\varepsilon = 1/k$, where k is a positive integer, and assume that all p_i are multiples of $1/k$. If we wish to satisfy the conditions of Proposition 1, a natural choice for k is $N+1$.

Lemma 3. *An up iteration at node i can only increase the surplus of nodes other than i. Once a node has nonnegative surplus, it continues to do so for the rest of the algorithm. Nodes with negative surplus have the same price as they did at the outset of the algorithm.*

Proof. The first statement is a direct consequence of the form of Steps 2 and 3 of the up iteration. The second then follows because each up iteration cannot drive the surplus of node i below zero, and can only increase the surplus of adjacent nodes. For the same reasons, a node with negative surplus can never have been the subject of an up iteration, and so its price must be the same as at initialization, proving the third claim. □

Finiteness

We now prove that the ε-relaxation algorithm terminates finitely. Since we will be giving an exact complexity estimate in the next section, this proof is not strictly necessary. However, it serves to illuminate the workings of the algorithm without getting involved in excessive detail.

Proposition 2. *If problem* (MCF) *is feasible, the pure form of the ε-relaxation method terminates with (f, p) satisfying ε-CS, and with f being integer and primal feasible.*

Proof. Because prices are nondecreasing (Lemma 1), there are two possibilities: either (a) the prices of a nonempty subset N^∞ of N diverge to $+\infty$, or else (b) the prices of all nodes in N remain bounded from above.

Suppose that case (a) holds. Then the algorithm never terminates, implying that at all times there must exist a node with negative surplus which, by Lemma 3, must have a constant price. Thus, N^∞ is a strict subset of N. To preserve ε-CS, we must have after a sufficient number of iterations

$$f_{ij} = c_{ij} \quad \text{for all } (i,j) \in A \text{ with } i \in N^\infty,\ j \notin N^\infty, \tag{20a}$$

$$f_{ji} = b_{ji} \quad \text{for all } (j,i) \in A \text{ with } i \in N^\infty,\ j \notin N^\infty, \tag{20b}$$

while the sum of surpluses of the nodes in N^∞ is positive. This means that even with as much flow as arc capacities allow coming out of N^∞ to nodes $j \notin N^\infty$, and as little flow as arc capacities allow coming into N^∞ from nodes $j \notin N^\infty$, the total surplus $\sum \{g_i \mid i \in N^\infty\}$ of nodes in N^∞ is positive. It follows that there is no feasible flow vector, contradicting the hypothesis. Therefore case (b) holds, and all the node prices stay bounded.

We now show that the algorithm terminates. If that were not so, then there must exist a node $i \in N$ at which an infinite number of iterations are executed. There must also exist an adjacent ε^--balanced arc (j, i), or ε^+-balanced arc (i, j) whose flow is decreased or increased (respectively) by an integer amount during an infinite number of iterations. For this to happen, the flow of (j, i) or (i, j) must be increased or decreased (respectively) an infinite number of times due to iterations at the adjacent node j. This implies that the arc (j, i) or (i, j) must become ε^+-balanced or ε^--balanced from ε^--balanced or ε^+-balanced (respectively) an infinite number of times. For this to happen, the price of the adjacent node j must be increased by at least 2ε an infinite number of times. It follows that $p_j \to \infty$ which contradicts the boundedness of all node prices shown earlier. Therefore the algorithm must terminate. □

Degenerate price rises

Note that when the push list is empty, the price p_i of the current node may be raised at the end of an up iteration even when $g_i = 0$. We call any price rise performed at i when $g_i = 0$ *degenerate*. Such price rises can be viewed as optional, and do not affect the finiteness or complexity of the algorithm. It is possible to omit them completely, and halt the up iteration as soon as $g_i = 0$. However, our computational experience has shown that degenerate price rises are a good idea in practice. Similar price changes have proved useful in the RELAX family of algorithms.

The reasons for this are not entirely clear, but some insight is obtained if we view degenerate steps as an attempt to increase the size of the price increases. The complexity analysis of the next section suggests that the algorithm terminates faster when the price increases are as large as possible.

Partial iterations

Actually, it is not strictly necessary to approximately maximize the dual cost with respect to p_i. One can also construct methods that work by repeatedly selecting nodes with positive surplus and applying *partial* up iterations to them. A *partial up iteration* is the same as an up iteration, except that it is permitted to halt following any execution of Step 2, 3, or 4. Thus, such algorithms are not constrained to reducing g_i to zero before turning their attention to other nodes. It turns out that these algorithms retain the finiteness and most of the complexity properties of ε-relaxation. They become important when one analyzes synchronous parallel implementations of ε-relaxation.

Broadbanding

Another useful variation on the basic up iteration, which we call *broadbanding*, is due to Goldberg and Tarjan [30–32]. In our terminology, broadbanding amounts to redefining the admissible arcs to be those that are active and have $f_{ij} < c_{ij}$, along

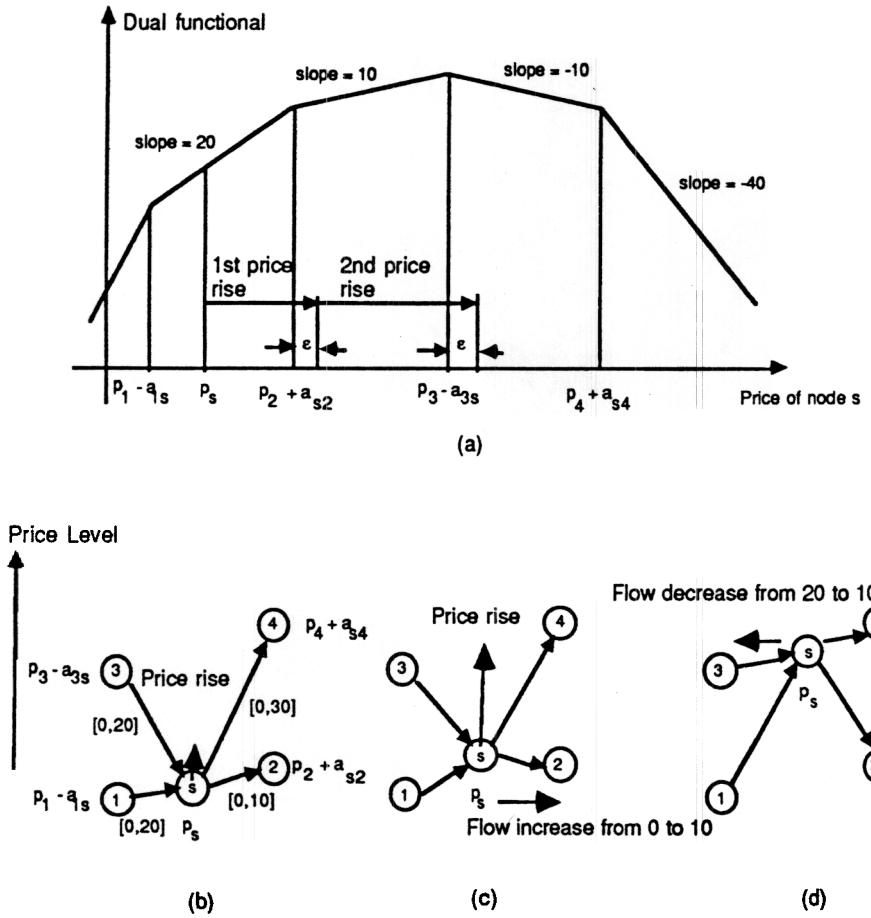

Fig. 6. Illustration of an up iteration involving a single node s with four incident arcs $(1, s)$, $(3, s)$, $(s, 2)$, and $(s, 4)$, with feasible arc flow ranges $[1, 20]$, $[0, 20]$, $[0, 10]$, and $[0, 30]$, respectively.

(a) Form of the dual functional along p_s for given values of p_1, p_2, p_3, and p_4. The breakpoints correspond to the levels of p_s for which the corresponding arcs become balanced. For values of p_s between two successive breakpoints there are no balanced arcs incident to node s. The corresponding slope of the dual cost is equal to the surplus g_s resulting when all active arc flows are set to their upper bounds and all inactive arc flows are set to their lower bounds; compare with (5).

(b) Illustration of a price rise of p_s from a value between the first two breakpoints to a value ϵ above the breakpoint at which $(s, 2)$ becomes balanced (Step 4).

(c) Price rise of p_s to a value ϵ above the breakpoint at which arc $(3, s)$ becomes balanced. When this is done, arc $(s, 2)$ has changed from ϵ^+-balanced to ϵ-active, and its flow has increased from 0 to 10, maintaining ϵ-CS.

(d) Step 3 of the algorithm reduces the flow of arc $(3, s)$ from 20 to 10, driving the surplus of node s to zero.

with those that are inactive and have $f_{ij} > b_{ij}$. Using ε-complementary slackness (6a-b), it follows that the admissible arcs consist of

$$(i,j) \text{ such that } f_{ij} < c_{ij} \text{ and } p_i - p_j \in (a_{ij}, a_{ij} + \varepsilon], \tag{21a}$$

$$(j,i) \text{ such that } f_{ji} > b_{ji} \text{ and } p_j - p_i \in [a_{ji} - \varepsilon, a_{ij}). \tag{21b}$$

We use the name *broadbanding* because arcs admissible for flow changes from their "start" nodes can have reduced costs anywhere in the band $[-\varepsilon, 0)$, whereas in regular ε-relaxation the reduced cost must be exactly $-\varepsilon$. A similar observation applies to admissible arcs eligible for flow changes from their "end" nodes.

Broadbanding makes it possible to drop the condition that ε divide all the arc costs and initial prices, yet still guarantee that all price rises are at least ε, which is useful in ε-scaling.

Down iterations

It is possible to construct a *down iteration* much like the above up iteration, which is applicable to nodes with $g_i < 0$, and reduces (rather than raises) p_i. Unfortunately, if one allows arbitrary mixing of up and down iterations, the ε-relaxation method may not even terminate finitely. Although experience with the RELAX methods [12, 17, 15] suggests that allowing a limited number of down iterations to be mixed with the up iterations might be a good idea in practice, our computational experiments with down iterations in ε-relaxation have not been conclusive.

4. Basic complexity analysis

We now commence a complexity analysis of ε-relaxation. We will develop a general analysis that will apply both the (pure) ε-relaxation algorithm we have already introduced, and to the scaled version we will discuss later.

The price bound $\beta(p)$

We develop the price bound $\beta(p)$, which is a function of the current price vector p, and serves to limit the amount of further price increases. For any path H, let $s(H)$ and $t(H)$ denote the start and end nodes of H, respectively, and let H^+ and H^- be the sets of arcs that are positively and negatively oriented, respectively, as one traverses the path from $s(H)$ to $t(H)$. We call a path *simple* if it is not a circuit and has no repeated nodes. For any price vector p and simple path H we define

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

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

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 arcs $(i,j) \in H^+$ minus the sum of $(p_i - p_j - a_{ij})$ over all arcs $(i,j) \in H^-$. For any flow f, we say that a simple path H is *unblocked with respect to* f if we have $f_{ij} < c_{ij}$ for all arcs $(i,j) \in H^+$, and we have $f_{ij} > b_{ij}$ for all arcs $(i,j) \in H^-$. In words, H is unblocked with respect to f if there is margin for sending positive flow along H (in addition to f) from $s(H)$ to $t(H)$ without violating the capacity constraints.

For any price vector p, and feasible flow f, define

$$D(p,f) = \max\{d_H(p) \mid H \text{ is a simple unblocked path with respect to } f\}. \tag{23}$$

In the exceptional case where there is no simple unblocked path with respect to f we define $D(p,f)$ to be zero. 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 f. Let

$$\beta(p) = \min\{D(p,f) \mid f \in \mathbb{Z}^A \text{ is feasible flow}\}. \tag{24}$$

There are only a finite number of values that $D(p,f)$ can take for a given p, so the minimum in (24) is attained for some f. The following lemma shows that $\beta(p)$ provides a measure of suboptimality of the price vector p. The computational complexity estimate we will obtain shortly is proportional to $\beta(p^0)$, where p^0 is the initial price vector.

Lemma 4. (a) *If, for some* $\gamma \geq 0$, *there exists a feasible flow* f *satisfying* γ-CS *together with* p *then*

$$0 \leq \beta(p) \leq (N-1)\gamma. \tag{25}$$

(b) p *is dual optimal if and only if* $\beta(p) = 0$.

Proof. (a) For each simple path H which is unblocked with respect to f and has $|H|$ arcs we have, by adding the γ-CS conditions given by (6a-b) along H and using (22),

$$d_H(p) \leq |H|\gamma \leq (N-1)\gamma, \tag{26}$$

and the result follows from (23) and (24).

(b) If p is optimal then it satisfies complementary slackness together with some primal optimal vector f, so from (26) (with $\gamma = 0$) we obtain $\beta(p) = 0$. Conversely if $\beta(p) = 0$, then from (24) we see that there must exist a primal feasible f such that $D(p,f) = 0$. Hence $d_H(p) = 0$ for all unblocked simple paths H with respect to f. Applying this fact to single-arc paths H and using the definition (16) we obtain that f together with p satisfy complementary slackness. Hence p and f are optimal. □

Price rise lemmas

We have already established that $\beta(p)$ is a measure of the optimality of p that is intimately connected with ε-complementary slackness. We now show that $\beta(p)$ also places a limit on the amount that prices can rise in the course of the ε-relaxation algorithm. Corollary 3.1 of [8] gives such a limit for the unscaled algorithm, but a more powerful result is required for the analysis of scaling methods. The first such result is contained in Lemmas 4 and 5 of [30], but does not use a general suboptimality measure like $\beta(p)$. The following lemma combines the analysis of [30] with that of [8], and is useful in both the scaled and unscaled cases.

Lemma 5. *If* (MCF) *is feasible, the number of price increases at each node is* $O(\beta(p^0)/\varepsilon + N)$.

Proof. Let (f, p) be a vector pair generated by the algorithm prior to termination, and let f^0 be a flow vector attaining the minimum in the definition (24) of $\beta(p^0)$. The key step is to consider $y = f - f^0$, which is a (probably not capacity-feasible) flow giving rise to the same surpluses $\{g_i, i \in N\}$ as f. If $g_t > 0$ for some node t, there must exist a node s with $g_s < 0$ and a simple path H with $s(H) = s$, $t(H) = t$, and such that $y_{ij} > 0$ for all $(i,j) \in H^+$ and $y_{ij} < 0$ for all $(i,j) \in H^-$. (This follows from the Conformal Realization Theorem [42, p. 104]. See also [25].)

By the construction of y, it follows that H is unblocked with respect to f^0. Hence, from (23) we must have $d_H(p^0) \leq D(p^0, f^0) = \beta(p^0)$, and by using (22),

$$p_s^0 - p_t^0 - \sum_{(i,j)\in H^+} a_{ij} + \sum_{(i,j)\in H^-} a_{ij} \leq \beta(p^0).$$

The construction of y also gives that the reverse of H must be unblocked with respect to f. Therefore, ε-complementary slackness (6a-b) gives $p_j \leq p_i - a_{ij} + \varepsilon$ for all $(i,j) \in H^+$ and $p_i \leq p_j + a_{ij} + \varepsilon$ for all $(i,j) \in H^-$. By adding these conditions along H we obtain

$$-p_s + p_t + \sum_{(i,j)\in H^+} a_{ij} - \sum_{(i,j)\in H^-} a_{ij} \leq |H|\varepsilon \leq (N-1)\varepsilon,$$

where $|H|$ is the number of arcs of H. We have $p_s^0 = p_s$ since the condition $g_s < 0$ implies that the price of s has not yet changed. Therefore, by adding (27) and (28) we obtain

$$p_t - p_t^0 \leq \beta(p^0) + (N-1)\varepsilon$$

throughout the algorithm for all nodes t with $g_t > 0$. From the assumptions and analysis of the previous section, we conclude that all price rises are by at least ε, so there are at most $\beta(p^0)/\varepsilon + (N-1)$ price increases at each node through the last time it has positive surplus. There may be one final degenerate price rise, so the total number of price rises is $\beta(p^0)/\varepsilon + N$ per node. □

In some cases, more information can be extracted from $f-f^0$ than in the above proof. For instance, Gabow and Tarjan [27] have shown that in assignment problems it is not only possible to bound the price of the individual nodes, but also the sum of the prices of all nodes with positive surplus. They use this refinement to construct an assignment algorithm with complexity $O(N^{1/2}A \log NC)$; however, the scaling subroutine used by this algorithm is a variant of the Hungarian method, rather than a dual coordinate step method. Ahuja and Orlin [2] have adapted this result to construct a hybrid assignment algorithm that uses the auction algorithm as a subroutine, but has the same complexity as the method of [27]. Their method switches to a variant of the Hungarian method when the number of nodes with positive surplus is sufficiently small. This bears an interesting resemblance to a technique used in the RELAX family of codes [12, 17, 46], which, under certain circumstances typically occurring near the end of execution, occasionally use descent directions corresponding to a more conventional primal-dual method.

Work breakdown

Now that a limit has been placed on the number of price increases, we must limit the amount of work associated with each price rise. The following basic approach to accounting for the work performed by the algorithm dates back to Goldberg and Tarjan's max-flow analysis [28, 29]. We define:

Scanning work to be the work involved in executing Step (4) of the up iteration — that is, computing new node prices and constructing the corresponding push lists. We also include in this category all work performed in removing items from push lists.

Saturating pushes are executions of Steps 2 and 3 of the up iteration in which an arc is set to its upper or lower flow bound (that is, $\delta = c_{ij} - f_{ij}$ in Step 2, or $\delta = f_{ji} - b_{ji}$ in Step 3).

Nonsaturating pushes are executions of Steps 2 and 3 that set an arc to a flow level strictly between its upper and lower flow bounds.

Limiting the amount of effort expended on the scanning and saturating pushes is relatively easy. From here on we will write β for $\beta(p^0)$ to economize on notation.

Lemma 6. *The amount of work expended in scanning is* $O(A(\beta/\varepsilon + N))$.

Proof. We already know that $O(\beta/\varepsilon + N)$ price rises may occur at any node. At any particular node i, Step 4 can be implemented so as to use $O(d(i))$ time, where $d(i)$ is the degree of node i. The work involved in removing elements from a push list built by Step 4 is similarly $O(d(i))$. Thus the total (sequential) work involved in scanning for all nodes is

$$O\left(\left[\sum_{i \in N} d(i)\right](\beta/\varepsilon + N)\right) = O(A(\beta/\varepsilon + N)). \quad \square \tag{30}$$

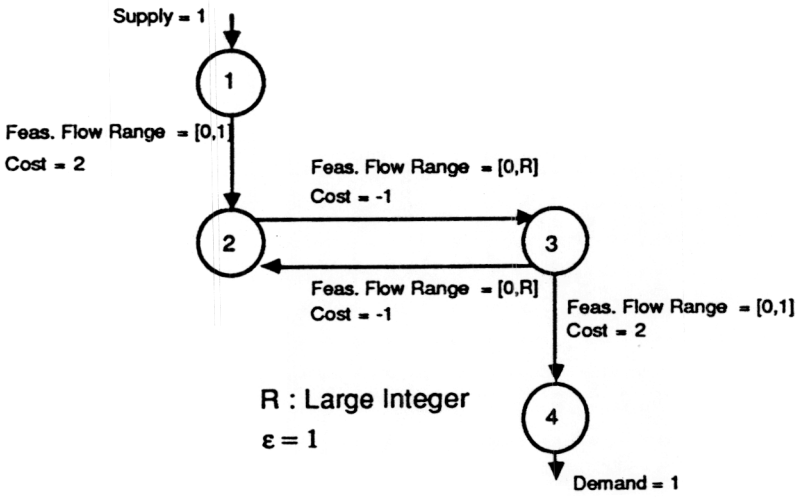

Fig. 7. Example showing the importance of keeping the admissible graph acyclic. Initially, we choose $f=0$, $p=0$, which do satisfy ε-complementary slackness, but imply a cyclic initial admissible graph. The algorithm will push one unit of flow R times around the cycle 2-3-2, taking $\Omega(R)$ time.

Lemma 7. *The amount of work involved in saturating pushes is also* $O(A(\beta/\varepsilon+N))$.

Proof. Each push (saturating or not) requires $O(1)$ time. Once a node i has performed a saturating push on an arc (i,j) or (j,i), there must be a price rise of at least 2ε by the node j before another push (necessarily in the opposite direction) can occur on the arc. Therefore, $O(\beta/\varepsilon+N)$ saturating pushes occur on each arc, for a total of $O(A(\beta/\varepsilon+N))$ work. □

Node ordering and the sweep algorithm

The main challenge in the theoretical analysis of the algorithm is containing the amount of work involved in *nonsaturating* pushes. There is a possibility of *flow looping*, in which a small amount of flow is "pushed" repeatedly around a cycle of very large residual capacity. Figure 7 illustrates that this can in fact happen. As we shall see, the problem can be avoided if the admissible graph is kept acyclic at all times. One way to assure this is by having $\varepsilon<1/N$. In that case, one can easily prove that the admissible graph must be acyclic by an argument similar to Proposition 1. However, we also have the following:

Lemma 8. *If the admissible graph is initially acyclic, it remains so throughout the executions of the ε-relaxation algorithm.*

Proof. All "push" operations (executions of Steps 2 and 3) can only remove arcs from the admissible graph; only price rises can insert arcs into the graph. Note also that in Lemma 1, we proved that when arcs are inserted, they are all directed out

of the node i at which the price rise was executed. Consequently, no cycle can pass through any of these arcs. □

Thus, it is only necessary to assure that the *initial* admissible graph is acyclic.

If it is acyclic, the admissible graph has a natural interpretation as a partial order on the node set N. A node i is called a *predecessor* of node j in this partial order if there is a directed path from i to j in the admissible graph. If i is a predecessor of j, then j is *descendant* of i. Each push operation moves surplus from one node to one of its immediate descendents, and surplus only moves "down" the admissible graph in the intervals between price changes.

The key to controlling the complexity of nonsaturating pushes is the interaction between the order in which nodes are processed and the order imposed by the admissible graph. The importance of node ordering was originally recognized in the max-flow work of [28] and [29], but the particular ordering used there does not work efficiently in the minimum-cost flow context.

To proceed with the analysis, we must first prohibit partial up iterations: every up iteration drives the surplus of its node to zero. Secondly, we assume that the algorithm is operated in *cycles*. A *cycle* is a set of iterations in which all nodes are chosen once in a given order, and an up iteration is executed at each node having positive surplus at the time its turn comes. The order may change from one cycle to the next.

A simple possibility is to maintain a fixed node order. The *sweep implementation*, given except for some implementation details in [8], is a different way of choosing the order, which is maintained in a linked list. Every time a node i changes its price, it is removed from its present list position and placed at the head of the list (this does not change the order in which the remaining nodes are taken up in the current cycle; only the order for the subsequent cycle is affected). We say that a given (total) node order is *compatible* with the order imposed by the admissible graph if no node appears before any of its predecessors.

Lemma 9. *If the initial admissible graph is acyclic and the initial node order is compatible with it, then the order maintained by the sweep implementation is always compatible with the admissible graph.*

Proof. By induction over the number of flow and price change operations. Flow alterations only delete arcs from the admissible graph, so they preserve compatibility. After a price rise at node i, i has no predecessors (by the proof of Lemma 1), hence it is permissible to move it to the first position. So price rises also preserve compatibility. □

Lemma 10. *Under the sweep implementation, if the initial node order is acyclic and the initial node order is compatible with it, then the maximum number of cycles is* $O(N(\beta/\varepsilon + N))$.

Proof. 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 increase takes place, all nodes in N^0 remain in N^0, and the execution of a complete up iteration at a node $i \in N^+$ moves i from N^+ to N^0. If no node changed price during a cycle, then all nodes of N^+ will be added to N^0 by the end of the cycle, implying that the algorithm terminates. Therefore there will be a node price change during every cycle except possibly for the last cycle. Since the number of price increases per node is $O(\beta/\varepsilon + N)$, this leads to an estimate of a total of $O(N(\beta/\varepsilon + N))$ cycles. □

Lemma 11. *Under the same conditions as Lemma* 10, *the total complexity of nonsaturating pushes is* $O(N^2(\beta/\varepsilon + N))$.

Proof. Nonsaturating pushes necessarily reduce the surplus of the current node i to zero, so there may be at most one of them per up iteration. There are less than N iterations per cycle, giving a total of $O(N^2(\beta/\varepsilon + N))$ possible nonsaturating pushes, each of which takes $O(1)$ time. □

Figure 8 depicts the sweep implementation.

Proposition 3. *Under the sweep implementation, if the initial admissible graph is acyclic and the initial node order is compatible with it, then the total complexity of the sweep implementation is* $O(N^2(\beta/\varepsilon + N))$.

Proof. Combining the results of Lemma 6, 7 and 11, we find that the dominant term is $O(N^2(\beta/\varepsilon + N))$, corresponding to the nonsaturating pushes (since we assume at most one arc in each direction between any pair of nodes, $A = O(N^2)$). The only other work performed by the algorithm is in maintaining the linked list, which involves only $O(1)$ work per price rise, and scanning down this list in the

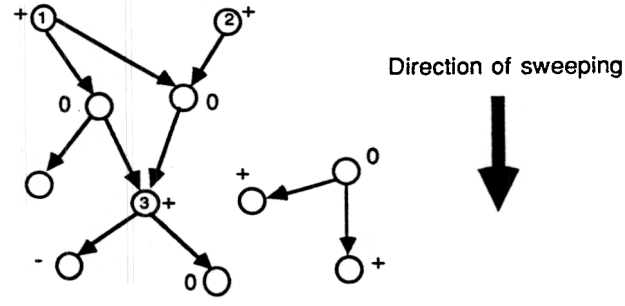

Fig. 8. Illustration of the admissible graph. A "+" (or "−" or "0") indicates a node with positive (or negative or zero) surplus. The algorithm is operated so that the admissible graph is acyclic at all times. The sweep implementation, based on the linked list data structure, processes high ranking nodes (such as nodes 1 and 2) before low ranking nodes (such as node 3).

course of each cycle, which involves $O(N)$ work per cycle. As there are $O(N^2(\beta/\varepsilon + N))$ price rises and $O(N(\beta/\varepsilon + N))$ cycles, both these leftover terms work out to $O(N^2(\beta/\varepsilon + N))$. □

A straightforward way of meeting the conditions of Proposition 3 is to choose p arbitrarily and set $f_{ij} = c_{ij}$ for all active (as opposed to ε-active) arcs and $f_{ij} = b_{ij}$ for all inactive ones. Then there will be no admissible arcs, and the initial admissible graph will be trivially acyclic. The initial node order may then be chosen arbitrarily.

The above proof also gives insight into the complexity of the method when other orders are used. At worst, only one node will be added to N^0 in each cycle, and hence that there may be $\Omega(N)$ cycles between successive price rises. In the absence of further analysis, one concludes that the complexity of the algorithm is a factor of N worse.

An alternate approach is to eschew cycles, and simply maintain a data structure representing the set of all nodes with positive surplus. [32] shows that a broad class of implementations of this kind have complexity $O(NA(\beta/\varepsilon + N))$. (Actually, these results are embedded in a scaling analysis, but the outcome is equivalent.)

We now give an upper bound on the complexity of the pure (unscaled) ε-relaxation algorithm, using the sweep implementation. Suppose we set the initial price vector p^0 to zero and choose f so that there are initially no admissible arcs. Then a crude upper bound on β is NC, where C is the maximum absolute value of the arcs costs, as in Section 2. Letting $\varepsilon = 1/(N+1)$ to assure optimality upon termination, we get an overall complexity bound of $O(N^4C)$. Figure 9 demonstrates that the time taken by the method can indeed vary linearly with C, so the algorithm is exponential.

Note also that any upper bound β^* on β provides a means of detecting infeasibility: If the problem instance (MCF) is not feasible, then the algorithm may abort in Step 4 of some up iteration, or some group of prices may diverge to $+\infty$. If any price increases by more than $\beta^* + N\varepsilon$, then we may conclude that such a divergence is happening, and halt with a conclusion of infeasibility. Thus, the total complexity may be limited to $O(N^2(\beta^*/\varepsilon + N))$, even without the assumption of feasibility. NC is always a permissible value for β^*.

Application to maximum flow

For classes of problems with special structure, a better estimate of $\beta(p^0)$ may be possible. As an example, consider the max-flow problem formulation shown in Fig. 10. The artificial arc (t, s) connecting the sink t with the source s has cost coefficient -1, and flow bounds $b_{ts} = 0$ and $c_{ts} = \sum_{i \in N} c_{si}$. We assume that $a_{ij} = 0$ and $b_{ij} = 0 < c_{ij}$ for all other arcs (i, j), and that $s_i = 0$ for all i. We apply the ε-relaxation algorithm with initial prices and arc flows satisfying ε-complementary slackness, where $\varepsilon = 1/(N+1)$. The initial prices may be arbitrary, so long as there is an $O(1)$ bound on how much they differ. Then we obtain $d_H(p^0) = O(1)$ for all paths H, $\beta(p^0) = O(1)$, and an $O(N^3)$ complexity bound. Note we may choose any positive value for ε and

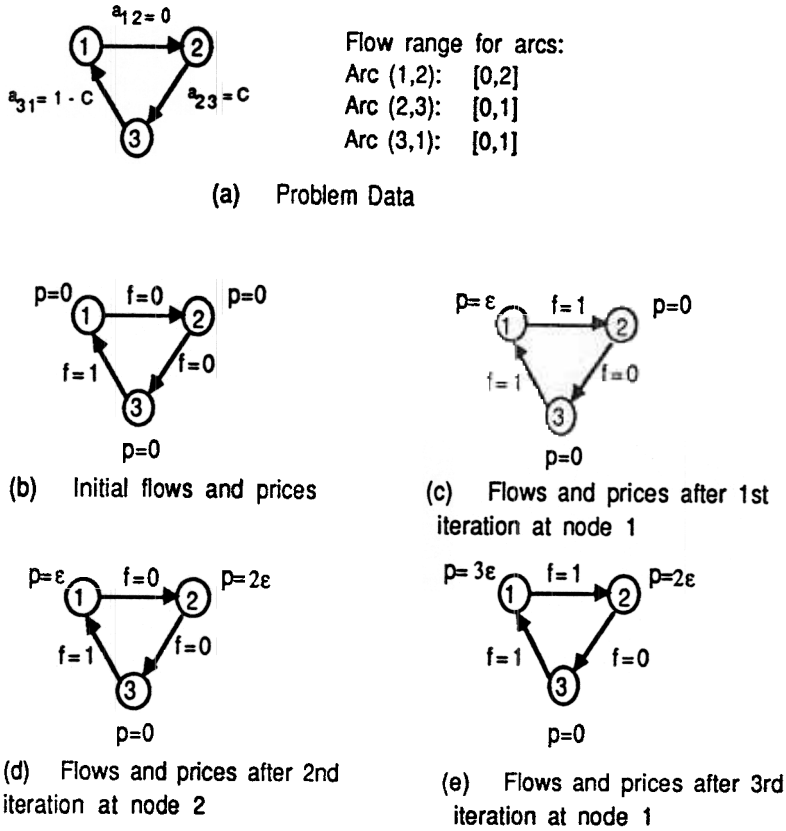

Fig. 9. Example showing that the computation required by the pure form of the ε-relaxation algorithm can be proportional to the cost-dependent factor C. Here, up iterations at node 1 alternate with up iterations at node 2 until the time when p_1 rises to the level $C-1+\varepsilon$ and arc $(3, 1)$ becomes ε^--balanced, so that a unit of flow can be pushed back along that arc. At this time, the optimal solution is obtained. Since prices rise by increments of no more than 2ε, the number of up iterations is $\Omega(C/\varepsilon)$.

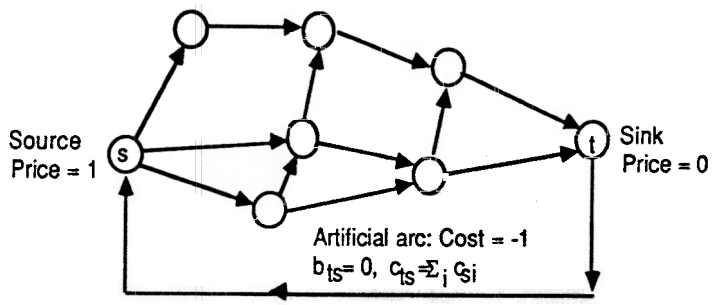

Fig. 10. Formulation of the max-flow problem.

negative values for a_{ts}, as long as $\varepsilon = -a_{ts}/(N+1)$ (more generally $\varepsilon = -a_{ts}/(1+\text{Largest number of arcs in a cycle containing } (t, s))$).

Applied like this to the maximum flow problem, ε-relaxation yields an algorithm resembling the maximum flow algorithm of [28–29], and having the same complexity. However, it has only one phase. The first phase of the procedure of [28–29] may in hindsight be considered an application of ε-relaxation with $\varepsilon = 1$ to the (infeasible) formulation of the maximum flow problem in which one considers all arcs costs to be zero, $s_s = -\infty$, and $s_t = +\infty$.

5. Scaling procedures

In general, some sort of *scaling procedure* [19, 24, 41] must be used to make the ε-relaxation algorithm polynomial. The basic idea is to divide the solution of the problem into a polynomial number of *subproblems* (also called *scales* or *phases*) in which ε-relaxation is applied, with $\beta(p^0)/\varepsilon$ being polynomial within each phase. The original analysis of this type, as we have mentioned, is due to Goldberg ([30] and, with Tarjan, [32]), who used ε-scaling. In order to be sure that all price rises are by $\Omega(\varepsilon)$ amounts, both these papers use the broadbanding variant of ε-relaxation as their principal subroutine (though they also present alternatives which are not dual coordinate step methods). Here, we will present an alternative cost scaling procedure, given in [4], that results in an overall complexity of $O(N^3 \log NC)$ and does not require broadbanding.

Cost scaling

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

$$a'_{ij} = (N+1)a_{ij} \quad \text{for all } (i,j). \tag{31}$$

If the pair (f', p') satisfies 1-complementary slackness (namely ε-complementary slackness with $\varepsilon = 1$) with respect to (SMCF), then clearly the pair

$$(f, p) = (f', p'/(N+1)) \tag{32}$$

satisfies $(N+1)^{-1}$-complementary slackness with respect to (MCF). Hence, if f' is feasible, it is optimal for (MCF) by Proposition 1. In the scaled algorithm, we seek a solution to (SMCF) obeying 1-complementary slackness.

Let

$$M = \lfloor \log_2(N+1)C \rfloor + 1 = O(\log(NC)). \tag{33}$$

In the scaled algorithm, we solve M subproblems, in each case using the sweep implementation of ε-relaxation. The mth subproblem is a minimum cost flow problem where the cost coefficient of each arc (i, j) is

$$a_{ij}(m) = \text{Trunc}(a'_{ij}/(2^{M-m})), \tag{34}$$

where Trunc($\cdot$) denotes integer rounding in the direction of zero, 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) one if the $(m+1)$st bit of the M-bit representation of a'_{ij} is a one and a'_{ij} is positive (negative). Note also that

$$a_{ij}(M) = a'_{ij}, \tag{35}$$

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

For each subproblem, we apply the unscaled version of the algorithm with $\varepsilon = 1$, yielding upon termination a pair $(f'(m), p'(m))$ satisfying 1-complementary slackness with respect to the cost coefficients $a_{ij}(m)$.

The starting price vector for the $(m+1)$st problem $(m = 1, 2, \ldots, M-1)$ is

$$p^0(m+1) = 2p'(m). \tag{36}$$

Doubling $p'(m)$ 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 $f^0(m+1)$ for the $(m+1)$st problem may be obtained from $f'(m)$ in any way that obeys 1-complementary slackness, keeps the admissible graph acyclic, and allows straightforward construction of a compatible node order. The simplest way to do this is to set

$$f^0_{ij}(m+1) = f'_{ij}(m) \quad \text{for all balanced arcs } (i,j), \tag{37a}$$

$$f^0_{ij}(m+1) = c_{ij} \quad \text{for all active arcs } (i,j), \text{ and} \tag{37b}$$

$$f^0_{ij}(m+1) = b_{ij} \quad \text{for all inactive arcs } (i,j). \tag{37c}$$

This procedure implies that the initial admissible graph for the $(m+1)$st problem has no arcs, and so an arbitrary node order (such as the one from the end of the last subproblem) may be used. A procedure that does not alter as many arc flows (and hence is likely to generate fewer nodes with nonzero surplus) is to set

$f^0_{ij}(m+1) = c_{ij}$ for all 1-active arcs (i,j),

$f^0_{ij}(m+1) = b_{ij}$ for all 1-inactive arcs (i,j),

$f^0_{ij}(m+1) = c_{ij}$ for all 1^+-active arcs (i,j) that were not admissible at the end of the previous phase,

$f^0_{ij}(m+1) = b_{ij}$ for all 1^--active arcs (i,j) that were not admissible at the end of the previous phase, and

$f^0_{ij}(m+1) = f'_{ij}(m)$ for all other arcs (i,j).

In this case, the arc set of the new admissible graph will be a subset of that prevailing at the end of subproblem m, hence the new graph will be acyclic. Furthermore, the node order at the end of phase m will be compatible with the new admissible graph, and may be used as the starting node order for phase $m+1$. For the *first* subproblem, however, there is no prior admissible graph, so the procedure (37a-c) must be used, and the initial node order can be arbitrary. The starting prices may be arbitrary so long as there is an $O(N)$ bound on how much they can differ.

Analysis

Using the analysis of Section 4, it is now fairly straightforward to find the complexity of the scaled form of the algorithm as outlined above.

Proposition 4. *The complexity of the scaled form of the ε-relaxation algorithm is* $O(N^3 \log NC)$.

Proof. Using Proposition 3 and $\varepsilon = 1$, the complexity of the scaled form of the algorithm is $O(N^2B + N^3M)$ where

$$B = \sum_{m=1}^{M} \beta_m(p^0(m)) \tag{38}$$

and $\beta_m(\cdot)$ is defined by (22)-(24) but with the modified cost coefficients $a_{ij}(m)$ replacing a_{ij} in (22). We show that

$$\beta_m(p^0(m)) = O(N) \quad \text{for all } m, \tag{39}$$

thereby obtaining an $O(N^3 \log NC)$ complexity bound, as $M = O(\log NC)$.

At the beginning of the first subproblem, we have

$$p_i - p_j = O(1),\ a_{ij}(1) = O(1) \quad \text{for all arcs } (i,j), \tag{40}$$

so we obtain $d_H(p^0(1)) = O(N)$ for all H, and $\beta_1(p^0(1)) = O(N)$. The final flow vector $f'(m)$ obtained from the m-th problem is feasible, and together with $p^0(m+1)$ it may be easily seen to satisfy 3-complementary slackness. It follows from Lemma 4(a) that

$$\beta_{m+1}(p^0(m+1)) \leq 3(N-1) = O(N). \tag{41}$$

It then follows that $B = O(NM)$, and the overall complexity is $O(N^3 \log NC)$. □

Practical experience with scaled ε-relaxation

Despite the good theoretical complexity bounds available for the scaled form of ε-relaxation and its relatives, dual coordinate algorithms have not yet proven themselves to be good performers in practice. Although nonsaturating pushes are the theoretical bottleneck in the algorithm, they present little problem in practice. We have observed that typically there are only a few flow alterations between

successive price rises. The real problem with the algorithm is the tendency of prices to rise at the theoretically minimum rate—by only ε or 2ε per price change. This is the phenomenon of *price haggling*. Essentially, the algorithm is following a "staircase" path in the dual (such as in Fig. 5), where the individual steps are very small.

Without scaling, the amount of price haggling can be exponential (as in Fig. 9), so scaling is clearly necessary to make ε-relaxation efficient. However, even with scaling, our computational experiments have shown that price haggling is still a serious difficulty. It often manifests itself in a prolonged "endgame" at the close of each subproblem, in which only a handful of nodes have positive surplus at any given time. Our experiments have also shown that degenerate price rises often cause a dramatic decrease in price haggling.

Even with scaling and degenerate steps, however, we have found ε-relaxation to be much slower than state-of-the-art sequential codes such as RELAX for large problems. We have not yet experimented with broadbanding and ε-scaling as opposed to cost scaling; although these techniques may offer some speed-up, we suspect it will not be dramatic. Also, the potential speed-up obtainable by a parallel implementation, as roughly indicated by the average number of nodes that simultaneously have positive surplus, appears to be only an order of magnitude or less. To make ε-relaxation algorithms viable, even on massively parallel machines, more work will need to be done to overcome price haggling.

6. The auction algorithm

The auction algorithm for the assignment problem, however, when combined with scaling, seems to have only limited difficulties with price haggling, and appears competitive with state-of-the-art codes even without any benefit from parallelism. Indeed, it has proved faster on a limited set of test problems.

Constructing auction from ε-relaxation

We now develop the auction algorithm as a variant of ε-relaxation. Note that the converse is also possible: by converting a minimum-cost flow problem to an assignment problem, and applying the auction algorithm, one may obtain a generic version of ε-relaxation. For a derivation of the auction method from first principles, refer to [10] and [11].

Consider a feasible assignment problem with n sources, n sinks, and an arbitrary set A of source-to-sink arcs. We say that source i is *assigned* to sink j if (i,j) has positive flow. All arcs are given capacity 1, so a flow change always sets an arc to its upper or lower bound, and all pushes are saturating. Thus, if one keeps track of the set of positive-surplus nodes such that the work of finding a node to iterate upon is always O(1), then the complexity of the pure ε-relaxation algorithm (using push lists) is reduced to $O(A(\beta/\varepsilon + N))$, regardless of the order in which nodes

are processed. Scaling therefore yields an $O(NA \log NC)$ algorithm. We now consider an algorithm in which up iterations are paired into "bids". Between bids (and also at initialization), only *source* nodes i can have positive surplus. Each bid does the following:

(I) Finds any unassigned source i (that is, one with positive surplus), and performs an up iteration at i.

(II) Takes the sink j to which i was consequently assigned, and performs an up iteration at j, *even if j has zero surplus.* If j has zero surplus, such an up iteration may just consist of a degenerate price rise. If the presence of an admissible arc on j's push list indicates that no price rise is possible, then this step takes just $O(1)$ time, aside from the work of removing inadmissible arcs from j's push list, which may be "charged" against earlier scanning steps.

More specifically, a bid by node i works as follows:

(a) Source node i sets its price to $p_j + a_{ij} + \varepsilon$, where j minimizes $p_k + a_{ik} + \varepsilon$ over all k for which $(i, k) \in A$. It then sets $f_{ij} = 1$, assigning itself to j.

(b) Node i then raises its price to $p_{j'} + a_{ij'} + \varepsilon$, where j' minimizes $p_k + a_{ik} + \varepsilon$ for $k \neq j$, $(i, k) \in A$.

(c) If sink j had a previous assignment $f_{i'j} = 1$, it breaks the assignment by setting $f_{i'j} := 0$ (one can show inductively that if this occurs, $p_j = p_{i'} - a_{i'j} + \varepsilon$).

(d) Sink j then raises its price p_j to

$$p_i - a_{ij} + \varepsilon = p_{j'} + a_{ij'} - a_{ij} + 2\varepsilon. \tag{42}$$

It is possible to rewrite the description of the bidding operation so that the prices of sinks do not explicitly appear. For compatibility with [10] and [11], we also formulate the assignment problem as a *maximization* by reversing the signs of all the a_{ij}. Let $\gamma = 2\varepsilon$, and define the *value* v_{ij} of a sink j to a source i to be $a_{ij} - p_j$. The rewritten bid iteration becomes

(1) Choose a person i who is unassigned.

(2) Find an object j^* that offers maximum value to i, that is

$$a_{ij^*} - p_{j^*} = \max_{(i,j)\in A}\{a_{ij} - p_j\}. \tag{43}$$

Also, find the best value offered by objects other than j^*, namely

$$w_{ij^*} = \max_{(i,j)\in A, j\neq j^*}\{a_{ij} - p_j\}. \tag{44}$$

(3) Compute the bid price

$$b_{ij^*} = a_{ij^*} - w_{ij^*} + \gamma, \tag{45}$$

and raise the price p_{j^*} of j^* to this level. Assign i to j^*, and break any prior assignment that j^* may have had.

What we have just described is the Gauss-Seidel or sequential version of the auction algorithm of [10, 11]. Those papers also show that several source nodes may place bids simultaneously. In that case, each sink node that receives more than one bid awards itself (provisionally) to the highest bidder. Hence the name "auction algorithm".

We may think of each node i as an agent who is trying to assign itself to an object j that comes within γ of offering the highest value to i. Once i has found the most desirable object j^*, it bids j^*'s price up to the *highest* level that still satisfies this criterion. In an actual auction involving real money, doing this would be foolish; however, we believe that this feature is instrumental in reducing price haggling and is precisely what makes the algorithm perform well in practice.

Push lists and complexity

If we implement the auction algorithm as a variation of ε-relaxation with a special node ordering scheme, as described above, then proper attention to push lists will insure an $O(A(\beta/\varepsilon + N))$ unscaled complexity. The only detail one must worry about is that up iterations begun at nodes with zero surplus (as in (II) above) do not add to the overall effort. The discussion in (II) above establishes this. Applying scaling then gives a complexity of $O(NA \log NC)$.

However, as in (a-d), it is possible to state the auction algorithm without reference to any of the source node prices p_i. We now present an implementation of auction that does not maintain source node prices, yet retains the complexity $O(NA \log NC)$.

Given any price vector p for the sink nodes, define an *artificial price* π_i of each source node i by

$$\pi_i = -\max_{(i,j)\in A}\{a_{ij} - p_j\}. \tag{46}$$

The reader may confirm that the prices π, p and the current flow (assignment) f always obey γ-complementary slackness. The reader may also refer to [11] for a proof that if f is feasible (that is, it is a complete assignment) and (f, π, p) satisfy γ-complementary slackness with $\gamma < 1/n = 2/N$, then f is optimal. This accords with Proposition 1 and the definition $\gamma = 2\varepsilon$.

Suppose there is a limit β^* on the amount that any single p_j can rise. From (a-d) above, all price rises are at least γ, so there are at most β^*/γ price rises at any sink, or—by (46)—at any source.

Each source node i maintains a *push list* consisting of all nodes except j^* that were tied for offering the value w_{ij^*} the last time i scanned its incident arcs. Along with each node is stored the price p'_j that prevailed for j *at the time the last scan was done.* The bids are performed as follows (note that, as in ε-relaxation, all prices are nondecreasing):

(1) Locate an unassigned source node i.

(2) Examine the elements (j, p'_j) of the push list of i, starting at the top. Discard any for which $p'_j < p_j$. Continue until reaching the end of the list, or the *second* element for which $p'_j = p_j$. If the end is reached, go to Step (4).

(3) Let j^* be the *first* element on the list for which $p'_j = p_j$. Discard the contents of the list up to, but not including, the *second* such element. Place a bid on j^* at price level $p_j + \gamma$, assigning i to j^* and breaking any prior assignment of j^*. Stop.

(4) Scan the incident arcs of i, determining an element j^* with maximum value, the next best value w_{ij^*}, as defined above, and all elements (other than j^*) tied at value level w_{ij^*}. Let the new push list of i be a list of all nodes j other than j^* tied at value level w_{ij^*}, coupled with their present prices. Submit a bid for j^* at price level b_{ij^*}, assigning i to j^* and breaking any prior assignment of j^*. Stop.

This method has complexity $O(A\beta^*/\gamma)$. We omit the details of the proof, but the key observation is that (4) can be performed in $O(d(i))$ time, and that between every two consecutive executions of (4) at a given node i, there must be an increase in the artificial price π_i of i. Placed in a scaling context where prices cannot rise by more than $\beta^*/\gamma = O(n)$ times per node in each subproblem, one can derive an overall complexity of $O(NA \log NC)$. However, it is doubtful that the overhead of keeping push lists in the auction algorithm is justified in practice. A simpler implementation that requires i to scan its incident arcs once per bid, whether or not there has been a change in π_i, can be shown to have complexity $\phi(N^3 \log NC)$.

Computational results

In this section we discuss limited computational experience with a public domain serial FORTRAN code called AUCTION, which implements the auction algorithm using ε-scaling. The initial sink prices were $p_j = \min_i a_{ij}$ for all j; this is a common choice for dual assignment algorithms. At the end of the kth subproblem, AUCTION checks the current assignment to see if it is optimal for the subproblem $k+1$, using the current prices π, p. If the current assignment does not obey ε-complementary slackness with π, p using the new value of ε, all assignments along ε-inactive arcs are deleted, and the auction is run again. After some experimentation, we found that we obtained the best performance by reducing ε by a factor of 5 between subproblems (using a factor other than 2 makes no significant difference in the complexity analysis). The initial value of ε was taken to be $nC/2$. AUCTION does not use push lists; every time a node bids, it simply scans all its incident arcs.

AUCTION implements a Gauss–Seidel version of the auction algorithm, in which only one node bids at a time. For computational results with a Jacobi version of AUCTION, which simulates all unassigned nodes bidding simultaneously, refer to [11]. The Gauss–Seidel version is somewhat faster, but not as amenable to parallel implementation.

Test problems were generated using the 1987 release version of the widely-used public domain generator NETGEN [34]. We generated problems with 800 to 12 000 total nodes, and an average node degree of 10. We compared the run times for AUCTION to those of the preexisting state-of-the-art public domain assignment code due to Jonker and Volgenant [33], also written in FORTRAN. This code implements a two-phase algorithm; the first phase is an initialization based on relaxation ideas of the type discussed in [12], while the second phase is a Hungarian method employing shortest path calculations. Both codes were run on a MicroVAX II CPU under the VMS 4.6 operating system.

Our results are summarized in Fig. 11. AUCTION appears faster to the Jonker-Volgenant code for problems having more than 1000 source nodes. Furthermore, the factor of superiority increases with problem size. Thus, the preliminary indication is that the auction algorithm is at least competitive with other serial methods for sparse assignment problems. This is consistent with the fact that the complexity $O(NA \log NC)$ of the auction algorithm is superior to the complexity $O(N^3)$ of Hungarian-type methods for sparse problems.

The results of Fig. 11 are typical of those obtained for sparse problems. When the problem to be solved is dense, the relative performance of the Jonker and Volgenant code improves markedly. The reason is that a substantial portion of the computation in the second phase of this code involves finding the minimum of a node-length array. The time to execute this operation is independent of problem density. We note that there are some unexpected features in AUCTION's performance on dense problems. We found that the algorithm with ε-scaling sometimes performs *worse* than the unscaled auction algorithm (where ε is fixed at $1/(n+1)$), particularly for large (!) C. It seems that for dense problems, price haggling typically

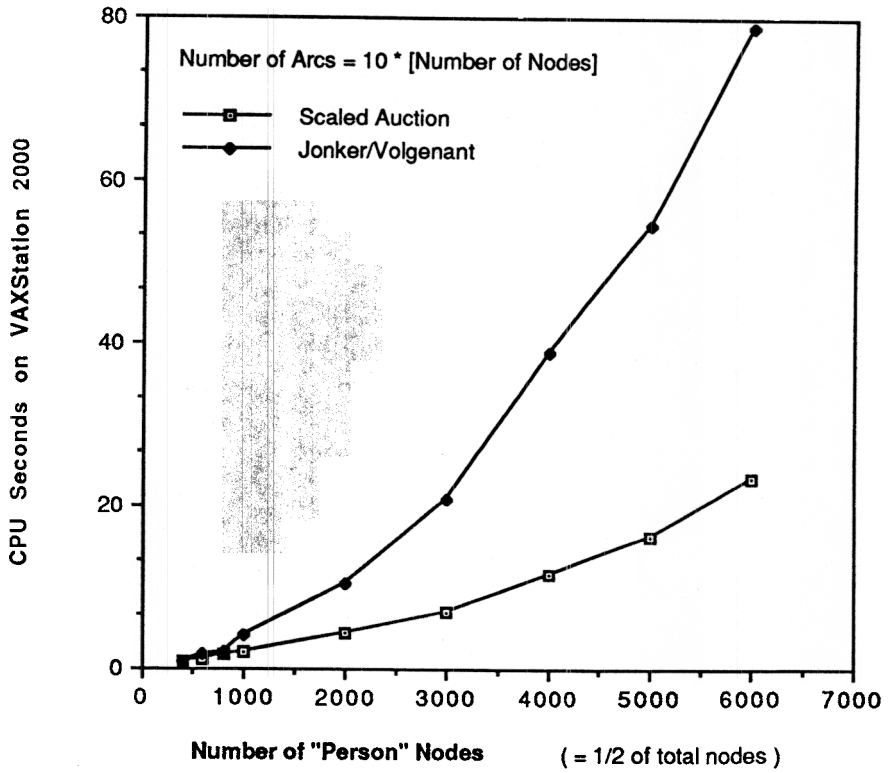

Fig. 11. Solution times for AUCTION and the Jonker-Volgenant assignment code on a VAXStation 2000, which uses the MicroVAX II CPU. All problems were created using NETGEN. Times do not include problem input or solution output.

becomes less of a difficulty when there is a lot of variation in the values of the arc cost coefficients. A related and somewhat surprising phenomenon is that the unscaled algorithm's performance for relatively small values of C may be worse than for large values of C.

Preliminary work on parallel implementations of the Jacobi auction algorithm is still in progress. While results by other researchers (quoted in [11]) for small numbers of processors appear encouraging, there are as yet no results for massively parallel environments. A central problem, as with all dual coordinate step algorithms, is that it is difficult to guarantee that a substantial fraction of the nodes will have positive surplus (that is, be unassigned) at any given time, and hence that a large number of processors can simultaneously be active. Indeed, experimental results have so far indicated that the average number of unassigned nodes is generally quite small. Thus, it may prove difficult to obtain massive speed-ups through parallelization.

7. Asynchronous implementation of ε-relaxation

So far as we know, nobody has been able to show how a true *theoretical* speed-up of either the auction or ε-relaxation algorithms may be obtained by a simple synchronous parallel implementation. The essential problem is that it is difficult to guarantee that more than one node will have positive surplus at any given time.

In this section, we will do something quite different: we demonstrate that there is a version of the ε-relaxation algorithm that converges even in a completely chaotic, asynchronous environment. Because the assumptions made in this model are so loose, it is not possible to come up with anything comparable to a complexity estimate. The real point is to show that the algorithm is resilient to the imperfections and inhomogeneities that may characterize some real-life distributed computing environments. The formulation involves a far more flexible type of asynchronism that can be obtained with the use of synchronizers [4]. Algorithmic convergence is often difficult to establish for chaotic models, but powerful results are now available to aid in this process [14, 18, 21]. The algorithm given here is more complex than a related algorithm for strictly convex arc costs [14], and requires a novel method of convergence proof.

We now return to the ordinary ε-relaxation method and assume that each node i is a processor that updates its own price and incident arc flows, and exchanges information with its "forward" adjacent nodes

$$F_i = \{j \mid (i, j) \in A\}, \tag{47}$$

and its "backward" adjacent nodes

$$B_i = \{j \mid (j, i) \in A\}. \tag{48}$$

The following distributed asynchronous implementation applies to both the pure algorithm and to the subproblems of the scaled method. The information available

at node i for any time t is as follows:

$p_i(t)$: The price of node i,

The price of node $j \in F_i \cup B_i$ communicated by j at some earlier time,

The estimate of the flow of arc (i,j), $j \in F_i$, available at node i at time t,

$f_{ji}(i,t)$: The estimate of the flow of arc (j,i), $j \in B_i$, available at node i at time t,

The estimate of the surplus of node i at time t given by

$$g_i(t) = \sum_{(j,i)\in A} f_{ji}(i,t) - \sum_{(i,2j)\in A} f_{ij}(i,t) - s_i. \tag{49}$$

A more precise description is possible, but for brevity we will keep our discussion somewhat informal. We assume that, for every node i, the quantities above do not change except possibly at an increasing sequence of times $t_0, t_1, \ldots$, with $t_m \to \infty$. At each of these times, generically denoted t, and at each node i, one of three events happens:

Event 1. Node i does nothing.

Event 2. Node i checks $g_i(t)$. If $g_i(t) \leq 0$, node i does nothing further. Otherwise node i executes either a complete or partial up iteration based on the available price and flow information

$$p_i(t), \quad p_j(i,t), j \in F_i \cup B_i, \quad f_{ij}(i,t), j \in F_i, \quad f_{ji}(i,t), j \in B_i$$

and accordingly changes

$$p_i(t), \quad f_{ij}(i,t), j \in F_i, \quad f_{ji}(i,t), j \in B_i$$

Event 3. Node i receives, from one or more adjacent nodes $j \in F_i \cup B_i$, a message containing the corresponding price and arc flow $(p_j(t'), f_{ij}(j,t'))$ (in the case $j \in F_i$), or $(p_j(t'), f_{ji}(j,t'))$ (in the case $j \in B_i$) stored at j at some earlier time $t' < t$. If

$$p_j(t') < p_j(i,t),$$

node i discards the message and does nothing further. Otherwise, node i stores the received value $p_j(t')$ in place of $p_j(i,t)$. In addition, if $j \in F_i$, node i stores $f_{ij}(j,t')$ in place of $f_{ij}(i,t)$ if

$$p_i(t) < p_j(t') + a_{ij} \quad \text{and} \quad f_{ij}(j,t') < f_{ij}(i,t)$$

and otherwise leaves $f_{ij}(i,t)$ unchanged; in the case $j \in B_i$, node i stores $f_{ji}(j,t')$ in

place of $f_{ji}(i, t)$ if

$$p_j(t') \geq p_i(t) + a_{ji} \quad \text{and} \quad f_{ji}(j, t') > f_{ji}(i, t)$$

and otherwise leaves $f_{ij}(i, t)$ unchanged. (Thus, in case of a balanced arc, the "tie" is broken in favor of the flow of the start node of the arc.)

Let T^i be the set of times for which an update by node i as in event 2 above is attempted, and let $T^i(j)$ be the set of times when a message is received at i from j as in event 3 above. We assume the following:

Assumption 1. Nodes never stop attempting to execute an up iteration, and receiving messages from all their neighbors, i.e., T^i and $T^i(j)$ have an infinite number of elements for all i and $j \in F_i \cup B_i$.

Assumption 2. Old information is eventually purged from the system, i.e., given any time t_k, there exists a time $t_m \geq t_k$ such that the time of generation of the price and flow information received at any node after t_m (i.e., the time t' in #3 above), exceeds t_k.

Assumption 3. For each i, the initial arc flows $f_{ij}(i, t_0)$, $j \in F_i$, and $f_{ji}(i, t_0)$, $j \in B_i$ are integer, and satisfy ε-CS together with $p_i(t_0)$ and $p_j(i, t_0)$, $j \in F_i \cup B_i$. Furthermore there holds

$$p_i(t_0) \geq p_i(j, t_0) \quad \text{for all } j \in F_i \cup B_i;$$

$$f_{ij}(i, t_0) \geq f_{ij}(j, t_0) \quad \text{for all } j \in F_i.$$

One set of initial conditions satisfying Assumption 3 but requiring little cooperation between processors is $p_j(i, t_0) \approx -\infty$ for i and $j \in F_i \cup B_i$, $f_{ij}(i, t_0) = c_{ij}$ and $f_{ij}(j, t_0) = b_{ij}$ for i and $j \in F_i$. Assumption 3 guarantees that for all $t \geq t_0$

$$p_i(t) \geq p_i(j, t'') \quad \text{for all } j \in F_i \cup B_i, \ t'' \leq t.$$

To see this, note that $p_i(t)$ is monotonically nondecreasing in t, and $p_i(j, t'')$ equals $p_i(t')$ for some $t' < t''$.

For all nodes i and times t, $f_{ij}(i, t)$ and $f_{ji}(i, t)$ are integer, and satisfy ε-CS together with $p_i(t)$ and $p_j(i, t)$, $j \in F_i \cup B_i$. This is seen from (50), the logic of the up iteration, and the rules for accepting information from adjacent nodes. Furthermore, for all i and $t \geq t_0$,

$$f_{ij}(i, t) \geq f_{ij}(j, t) \quad \text{for all } j \in F_i,$$

i.e., the start node of an arc has at least as high an estimate of arc flow as the end node. For a given $(i, j) \in A$, condition (51) holds initially by Assumption 3, and it

is preserved by up iterations at i since they cannot decrease $f_{ij}(i, t)$, while an up iteration at j cannot increase $f_{ij}(j, t)$. It can also be shown that (51) cannot be violated at the time of a message reception, but we omit the proof.

Once a node i gets nonnegative surplus $g_i(t) \geq 0$, it maintains a nonnegative surplus for all subsequent times. The reason is that an up iteration at i can at most decrease $g_i(t)$ to zero, while in view of the rules for accepting messages, a message exchange with an adjacent node j can only increase $g_i(t)$. Note also that from (51) we obtain

$$\sum_{i \in N} g_i(t) \leq 0 \quad \text{for all } t \geq t_0$$

This implies that, at any time t, there is at least one node i with negative surplus $g_i(t)$ if there is a node with positive surplus. This node i must not have executed any up iteration up to time t, and therefore its price $p_i(t)$ must still be equal to the initial price $p_i(t_0)$.

We say that the algorithm terminates if there is a time t_k such that for all $t \geq t_k$ we have

$$g_i(t) = 0 \qquad \text{for all } i \in N,$$

$$f_{ij}(i, t) = f_{ij}(j, t) \quad \text{for all } (i, j) \in A,$$

$$p_j(t) = p_j(i, t) \qquad \text{for all } j \in F_i \cup B_i.$$

Termination can be detected by using an adaptation of the protocol for diffusing computations of [23]. Our main result is:

Proposition 5. *If* (MCF) *is feasible and Assumptions* 1-3 *hold, the distributed, totally asynchronous version of the algorithm terminates.*

Proof. Suppose no up iterations are executed at any node after some time t^*. Then (53) must hold for large enough t. Because no up iterations occur after t^*, all the $p_i(t)$ must thenceforth remain constant, and Assumption 1, (50), and the message acceptance rules imply (55). After t^*, no flow estimates may change except by message reception. By (55), the nodes will eventually agree on whether each arc is active, inactive, or balanced. The message reception rules, (51), and Assumptions 1-2 then imply the eventual agreement on arc flows (54). (Eventually, the start node of each inactive arc will accept the flow of the end node, and the end node of a balanced or active arc will accept the flow of the start node.)

We now assume the contrary, i.e., that up iterations are executed indefinitely, and hence for every t there is a time $t' > t$ and a node i such that $g_i(t') > 0$. There are two possibilities: The first is that $p_i(t)$ converges to a finite value p_i for every i. In this case we assume without loss of generality that there is at least one node i at which an infinite number of up iterations are executed, and an adjacent arc (i, j)

whose flow $f_{ij}(i, t)$ is changed by an integer amount an infinite number of times with (i, j) being ε^+-balanced. For this to happen there must be a reduction of $f_{ij}(i, t)$ through communication from j an infinite number of times. This means that $f_{ij}(j, t)$ is reduced an infinite number of times which can happen only if an infinite number of up iterations are executed at j with (i, j) being ε^--balanced. But this is impossible since, when p_i and p_j converge, arc (i, j) cannot become both ε^+-balanced and ε^--balanced infinitely often.

The second possibility is that there is a nonempty subset of nodes N^∞ whose prices increase to ∞. It is seen then that there is at least one node that has negative surplus for all t, and therefore also a constant price. It follows that N^∞ is a strict subset of N. Since the algorithm maintains ε-CS, we have for all sufficiently large t that

$$f_{ij}(i, t) = f_{ij}(j, t) = c_{ij} \quad \text{for all } (i, j) \in A \text{ with } i \in N^\infty, j \notin N^\infty,$$

$$f_{ji}(i, t) = f_{ji}(j, t) = b_{ji} \quad \text{for all } (j, i) \in A \text{ with } i \in N^\infty, j \notin N^\infty.$$

Note now that all nodes in N^∞ have nonnegative surplus, and each must have positive surplus infinitely often. Adding (49) for all i in N^∞, and using both (51) and the above relations, we find that the sum of c_{ij} over all $(i, j) \in A$ with $i \in N^\infty$, $j \notin N^\infty$, plus the sum of s_i over $i \in N^\infty$ is less than the sum of b_{ji} over all $(j, i) \in A$ with $i \in N^\infty$, $j \notin N^\infty$. Therefore, there can be no feasible solution, violating the hypothesis. It follows that the algorithm must terminate. □

8. Conclusions

Coordinate step methods are based on a blend of classical nonlinear programming ideas of duality and coordinate ascent, and the notion of ε-complementary slackness, which has its roots in nondifferentiable optimization (see, for instance, [16]). The methods were motivated, starting with the auction algorithm, by the desire to massively parallelize the solution of network flow problems. However, they have yet to fulfill their promise in this regard, either analytically or computationally. The speed-up they provide in the limited parallel computational experimentation performed so far is not spectacular, and their parallel complexity has not yet been shown to be superior to their serial complexity. Their serial complexity *has* been shown to be very favorable with proper implementation; yet their performance has yet to match the theoretical promise. In particular, the ε-relaxation method has yet to be shown to approach the actual performance of the earlier (theoretically pseudopolynomial) relaxation methods [12, 17, 46]. The auction algorithm is the only member of the class which has been shown to be computationally competitive with existing serial methods (and probably superior for many types of assignment problems). There are as yet no published computational experimental results concerning coordinate step methods for max-flow problems. We should also add that

coordinate step methods are still recent, and not yet fully understood. Further research may substantially change the preceding assessment.

References

[1] R.K. Ahuja and J.B. Orlin, "A fast and simple algorithm for the maximum flow problem," *Operations Research* (to appear).

[2] R.K. Ahuja and J.B. Orlin, private communication, November 1987.

[3] R.K. Ahuja, J.B. Orlin and R.E. Tarjan, "Improved time bounds for the maximum flow problem," Sloan School Working Paper 1966-87, MIT, 1987.

[4] Awerbuch, B., "Complexity of network synchronization," *Journal of the ACM* 32 (1985) 804-823.

[5] D.P. Bertsekas and J. Eckstein, "Distributed asynchronous relaxation methods for linear network flow problems," Proceedings of International Federation of Automatic Control, Munich, July 1987.

[6] D.P. Bertsekas, "Distributed relaxation methods for linear network flow problems," *Proceedings of 25th IEEE Conference on Decision and Control*, Athens, Greece, 1986, pp. 2101-2106.

[7] D.P. Bertsekas, "Distributed asynchronous relaxation methods for linear network flow problems," LIDS Report P-1606, MIT, Sept. 1986.

[8] D.P. Bertsekas, "Distributed asynchronous relaxation methods for linear network flow problems," LIDS Report P-1606, MIT, revision of Nov. 1986.

[9] D.P. Bertsekas, "A distributed algorithm for the assignment problem," Unpublished LIDS Working Paper, MIT, March 1979.

[10] D.P. Bertsekas, "A distributed asynchronous relaxation algorithm for the assignment problem," *Proc. 24th IEEE Conference on Decision and Control*, Ft Lauderdale, FL, Dec. 1985, pp. 1703-1704.

[11] D.P. Bertsekas, "The auction algorithm: A distributed relaxation method for the assignment problem," *Annals of Operations Research* 14 (1988) 105-123.

[12] D.P. Bertsekas, "A unified framework for primal-dual methods in minimum cost network flow problems," *Mathematical Programming* 32 (1985) 125-145.

[13] D.P. Bertsekas and D.P. Castañon, "The auction algorithm for transportation problems," unpublished manuscript, November 1987.

[14] D.P. Bertsekas and D. El Baz, "Distributed asynchronous relaxation methods for convex network flow problems," *SIAM Journal on Control and Optimization* 25 (1987) 74-85.

[15] D.P. Bertsekas, P. Hossein and P. Tseng, "Relaxation methods for network flow problems with convex arc costs," *SIAM Journal on Control and Optimization* 25 (1987) 1219-1243.

[16] D.P. Bertsekas and S.K. Mitter, "A descent numerical method for optimization problems with nondifferentiable cost functionals," *SIAM Journal on Control and Optimization* 11 (1973) 637-652.

[17] D.P. Bertsekas and P. Tseng, "Relaxation methods for minimum cost ordinary and generalized network flow problems," *Operations Research* 36 (1988) 93-114.

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

[19] R.G. Bland and D.L. Jensen, "On the computational behavior of a polynomial-time network flow algorithm," Tech. Report 661, School of Operations Research and Industrial Engineering, Cornell University, June 1985.

[20] Y. Censor and A. Lent, "An iterative row-action method for interval convex programming," *Journal of Optimization Theory and Applications* 34 (1981) 321-352.

[21] D. Chazan and W. Miranker, "Chaotic relaxation," *Linear Algebra and its Applications* 2 (1969) 199-222.

[22] R.W. Cottle and J.S. Pang, "On the convergence of a block successive over-relaxation method for a class of linear complementarity problems," *Mathematical Programming Study* 17 (1982) 126-138.

[23] E.W. Dijkstra and C.S. Sholten, "Termination detection for diffusing computations," *Information Processing Letters* 11 (1980) 1-4.

[24] J. Edmonds and R.M. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems," *Journal of the ACM* 19 (1972) pp. 248-264.

[25] L.R. Ford and D.R. Fulkerson, *Flows in Networks* (Princeton University Press, Princeton NJ, 1962).

[26] H.N. Gabow and R.E. Tarjan, "Faster scaling algorithms for graph matching," unpublished manuscript, 1987.
[27] H.N. Gabow and R.E. Tarjan, "Faster scaling algorithms for network problems," unpublished manuscript, July 1987.
[28] A.V. Goldberg, "A new max-flow algorithm," Tech. Mem. MIT/LCS/TM-291, Laboratory for Computer Science, MIT, 1985.
[29] A.V. Goldberg and R.E. Tarjan, "A new approach to the maximum flow problem," Proc. 18th ACM STOC, 1986, pp. 136–146.
[30] A.V. Goldberg, "Solving minimum-cost flow problems by successive approximations," extended abstract, submitted to *STOC 87*, Nov. 1986.
[31] A.V. Goldberg, "Efficient graph algorithms for sequential and parallel computers," Tech. Report TR-374, Laboratory for Computer Science, MIT, Feb. 1987.
[32] A.V. Goldberg and R.E. Tarjan, "Solving minimum cost flow problems by successive approximations," Proc. 19th ACM STOC, May 1987.
[33] R. Jonker and A. Volgenant, "A shortest augmenting path algorithm for dense and sparse linear assignment problems," *Computing* V. 38 (1987) 325–340.
[34] D. Klingman, A. Napier and J. Stutz, "NETGEN—A program for generating large scale (un)capacitated assignment, transportation, and minimum cost flow network problems," *Management Science* 20 (1974) pp. 814–822.
[35] D.G. Luenberger, *Linear and Nonlinear Programming* (Addison-Wesley, Reading, MA, 1984).
[36] A. Ohuchi and I. Kaji, "Lagrangian dual coordinatewise maximization algorithm for network transportation problems with quadratic costs," *Networks* 14 (1984).
[37] J.B. Orlin, "Genuinely polynomial simplex and non-simplex algorithms for the minimum cost flow problem," Working Paper No. 1615-84, Sloan School of Management, MIT, Dec. 1984.
[38] J.S. Pang, On the convergence of dual ascent methods for large-scale linearly constrained optimization problems, Univeristy of Texas at Dallas, unpublished manuscript, 1984.
[39] C.H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity* (Prentice-Hall, Englewood Cliffs, NJ, 1982).
[40] B.T. Polyak, "Minimization of unsmooth functions," *USSR Computational Mathematics and Mathematical Physics* 9 (1969) pp. 14–29.
[41] H. Rock, "Scaling techniques for minimal cost network flows," in V. Page, ed., *Discrete Structures and Algorithms* (Carl Hansen, Munich, 1980).
[42] R.T. Rockafellar, *Network Flows and Monotropic Programming* (J. Wiley, NY, 1984).
[43] R.T. Rockafellar, *Convex Analysis* (Princeton Univ. Press, Princeton, NJ, 1970).
[44] T.E. Stern, "A class of decentralized routing algorithms using relaxation," *IEEE Transactions on Communication* 25(10) (1977).
[45] E. Tardos, "A strongly polynomial minimum cost circulation algorithm," *Combinatorica* 5 (1985) 247–255.
[46] P. Tseng, "Relaxation methods for monotropic programming problems," PhD Thesis, Dept. of Electrical Engineering and Computer Science, MIT, May 1986.
[47] S.A. Zenios and R.A. Lasken, "Nonlinear network optimization on a massively parallel connection machine," Report 87-08-03, Decision Sciences Department, The Wharton School, University of Pennsylvania, 1987.
[48] S.A. Zenios and J.M. Mulvey, "Relaxation techniques for strictly convex network problems," *Annals of Operations Research* 5 (1986).