# Dual Coordinate Ascent Methods for Linear Network Flow Problems*

by

**Dimitri P. Bertsekas and Jonathan Eckstein**
**Laboratory for Information and Decision Systems**
**and Operations Research Center**
**Massachusetts Institute of Technology**
**Cambridge, Mass. 02139**

## Abstract

We review a class of recently-proposed linear-cost network flow methods which are amenable to parallel implementation. All the methods in the class use the notion of $\epsilon$-complementary slackness, and most do not explicitly manipulate any "global" objects such as paths, trees, or cuts. Interestingly, these methods have also stimulated a large number of new *serial* computational complexity results. We develop the basic theory of these methods and present two specific methods, the *$\epsilon$-relaxation* algorithm for the minimum-cost flow problem, and the *auction* algorithm for 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 $\epsilon$-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.

---

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

# 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 ascent methods*. Their appearance has also stimulated a flurry of advances in *serial* computational complexity results for network optimization problems.

Another feature of this class of 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.

In this paper, we will concentrate on local algorithms, since they are the ones which hold the promise of efficient parallel implementation, and show how they can be regarded as coordinate ascent or relaxation methods in an appropriately-formulated dual problem. We will then take a detailed look at what is perhaps the generic algorithm of the class, the *ε-relaxation* algorithm, and give both scaled and unscaled complexity results for it. We will also consider a related algorithm for assignment problems, which we call *auction*, along with its scaled and unscaled complexities. We discuss the practical performance of these algorithms, based on preliminary experimentation. Finally, we will present an implementation of ε-relaxation that works in a completely asynchronous, chaotic environment.

# 2. Basic Concepts

## 2.1. Duality

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 ([36], Ch.7) may be written

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

subject to

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

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

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) ∈ **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 ∈ **N**, we can write the corresponding Lagrangian function as

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

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

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

subject to no constraint on p,

with the dual functional q given by

$$q(p) = \min_f \{L(f, p) \mid b_{ij} \le f_{ij} \le c_{ij},\ (i, j) \in A\}$$

$$= \sum_{(i,j)\in A} q_{ij}(p_i - p_j) + \sum_{i\in N} s_i p_i \qquad (5a)$$

where

$$q_{ij}(p_i - p_j) = \min_{f_{ij}} \{(a_{ij} + p_j - p_i)f_{ij} \mid b_{ij} \le f_{ij} \le c_{ij}\} \qquad (5b)$$

The function $q_{ij}$ is shown in Figure 1. This formulation of the dual problem is consistent with conjugate duality frameworks [39], [40] but can also be obtained via linear programming duality theory [32], [36]. 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*.

The results of this paper admit extension to the case where some or all of the $c_{ij}$ are infinite, introducing constraints into the dual problem. We omit these extensions in the interest of brevity.

## 2.2. Primal-Dual Coordinate Ascent: the Up and Down Iterations

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. However, the nondifferentiability of the dual function q presents special difficulties.

Because of these difficulties, the algorithms we will examine in this paper are in fact *primal-dual* methods, in that they maintain not only a vector of prices p, but also a capacity-feasible flow f such that f and p jointly satisfy (perhaps approximately) the *complementary slackness* conditions

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

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

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 ([39] and [40]), there is a useful interpretation of the complementary slackness conditions (6a-b). Referring to Figure 1, the complementary slackness conditions on (i, j) and the capacity constraint $b_{ij} \le f_{ij} \le c_{ij}$ 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 the *surplus* of node i, defined to be

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

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 obvious 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

| | | | |
|---|---|---|---|
| *Inactive* | if | $p_i < a_{ij} + p_j$ | (8a) |
| *Balanced* | if | $p_i = a_{ij} + p_j$ | (8b) |
| *Active* | if | $p_i > a_{ij} + p_j$ . | (8c) |

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

| | | |
|---|---|---|
| $f_{ij} = b_{ij}$, | if (i, j) active | (9a) |
| $b_{ij} \le f_{ij} \le c_{ij}$, | if (i, j) balanced | (9b) |
| $f_{ij} = c_{ij}$, | if (i, j) active . | (9c) |

Figure 2 displays the form of the dual function along a single price coordinate $p_i$. The breakpoints along the curve correspond to points where one or more arcs incident to node i are balanced. 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).

It is now clear how to maintain a pair (f, p) satisfying complementary slackness while altering a single dual variable $p_i$. Each time $p_i$ passes through a breakpoint, one simply sets each arc incident to i to its upper flow bound if it has become active, or to its lower flow bound if it has become inactive. Often, however, it is useful to perform a somewhat more involved calculation that takes advantage of the supergradient properties of the surplus $g_i$. This calculation also supplants any direct computation the directional derivatives of q. Suppose that we have (f, p) satisfying complementary slackness, some node i for which $g_i > 0$, and we are at a breakpoint in the dual cost. Since $g_i$ is a supergradient of $q_i(p_i; p)$, decreasing $p_i$ must decrease the dual objective value q, so either the current value of $p_i$ is optimal or it should be increased. We then try to decrease the surplus $g_i$ of i by "pushing" flow on the balanced arcs incident to i — that is, increasing the flow on balanced outgoing arcs and decreasing the flow on incoming balanced arcs, to the extent permitted by the capacity constraints (2). If the surplus can be reduced to zero in this way , we conclude that $0 \in \partial q_i(p_i; p)$, and hence that the current value of $p_i$ is optimal. Otherwise, we set all outgoing balanced arcs to their maximum flow, and all incoming balanced arcs to their minimum. The resulting surplus $g_i$ is then the minimal member of $\partial q_i(p_i; p)$, and hence (by the concavity of

q) the directional derivative of q in the positive $p_i$ direction. Consequently, we may improve the dual objective by increasing $p_i$ until we reach the next breakpoint, that is, until another arc becomes balanced. At this point, we repeat the entire procedure, stopping only when we obtain $g_i = 0$.

What we have just described is the basic outline of what we call the *up iteration*, which is central to all dual coordinate ascent methods. It maximizes q along the $p_i$ coordinate by increasing $p_i$, while maintaining the capacity constraints and the complementary slackness of f and p. It reduces the surplus of the node i to zero. There is also an entirely analogous *down iteration* that applies to nodes with $g_i < 0$, reduces the variable $p_i$, and increase $g_i$ to zero.

## 2.3. Jamming and the RELAX Approach

At this point, it may seem appealing to consider maximizing the dual function by starting with some arbitrary pair (f, p) satisfying (9a-c) and repeatedly applying up and down iterations until all nodes have zero surplus. It would then follow that the final f and p obtained would be primal and dual optimal, respectively. Unfortunately, as shown in Figure 3, a nondifferentiable function such as q, although continuous and concave, may have suboptimal points where it cannot be improved by either increasing or decreasing any *single* variable. If this naive algorithm were to encounter such a point, it would perform an infinite sequence of changes to the flow $f_{ij}$ without ever halting or changing the dual prices p. We call this phenomenon *jamming*.

One way of avoiding the jamming problem is embodied in the RELAX family of serial computer codes (see [11], [13], [43]). 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, they are not particularly suitable for massively parallel environments because of the difficulties of coordinating simultaneous multiple-node price change and labelling 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 [41], [19], [35], [21], and [33]. [45] contains computational results on a simulated parallel architecture, and [44] results on an actual parallel machine. [14] and [15] contain convergence proofs.

## 2.4. The Auction Approach

A different, more radical approach to the jamming problem is to allow small price changes, say by some amount $\varepsilon$, even if they worsen the dual cost. This idea dates back to the auction algorithm ([8],[9],[10]), a procedure for the assignment (bipartite matching) problem that actually predates the RELAX family of algorithms. In this algorithm, one considers the nodes on one side of the bipartite graph to be "people" or agents placing bids for the "objects" represented 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).

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 *$\varepsilon$-relaxation*. First, we must introduce the concept of *$\varepsilon$-complementary slackness*.

## 2.5. $\varepsilon$–Complementary Slackness

The *$\varepsilon$-complementary slackness* conditions are obtained by "softening" the two inequalities of the conventional complementary slackness conditions (6a-b) by an amount $\varepsilon \geq 0$, yielding:

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

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

The "kilter diagrams" of Figure 4 display the relationship imposed between $p_i - p_j$ and $f_{ij}$ by both $\varepsilon$-complementary slackness and regular complementary slackness. We say that the arc (i, j) is

$\varepsilon$-Inactive if $p_i < a_{ij} + p_j - \varepsilon$ (11a)

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

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

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

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

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

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

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

$$f_{ij} = c_{ij} \qquad \text{if} \qquad (i, j) \text{ is } \varepsilon\text{-active} \ . \tag{12c}$$

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

$$\sum_{(i,j)\in Y^+} a_{ij} \ - \sum_{(i,j)\in Y^-} a_{ij} < 0 \tag{13a}$$

$$f_{ij} < c_{ij} \qquad \text{for } (i, j) \in Y^+ \tag{13b}$$

$$b_{ij} < f_{ij} \qquad \text{for } (i, j) \in Y^-. \tag{13c}$$

Using (10a-b), we have

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

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

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

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

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

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

$$\varepsilon < \min_{\text{All directed cycles } Y} \left\{ \frac{-\text{Length of } Y}{\text{Number of arcs in } Y} \ \middle| \ \text{Length of } Y < 0 \right\} , \tag{15}$$

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 $d \ge 1$.

The notion of ε-complementary slackness was used in [8], [9], and introduced more formally in [13], [14]. It was also used in the analysis of [42] (Lemma 2.2) in the special case where the flow vector f is feasible. 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 [39]) of the primal space.

## 2.6. The Basics of the ε–Relaxation Method

We are now ready to describe the outlines of the ε-relaxation algorithm. It starts with any integer flow f and prices p satisfying ε-complementary slackness. Such a pair may be constructed by choosing p arbitrarily and then constructing f so as to obey (12a-c). The algorithm then repeatedly selects nodes i with positive surplus ($g_i > 0$) and performs up iterations at them. Unlike the basic up iteration described above, though, these iterations set $p_i$ to a value ε *above* the maximizer of the dual cost with respect to $p_i$. The flow is adjusted to maintain integrality and ε-complementary slackness, but not necessarily regular complementary slackness. As we shall prove below, this process will eventually drive all the nodes' surpluses to zero, resulting in a pair (f, p) satisfying the conditions of Proposition 1 (presuming $\varepsilon < 1/N$). It avoids jamming by following paths such as those depicted in figure 5. If $\varepsilon \geq d/N$, where d is the greatest common divisor of the arc costs, the algorithm will still terminate with a feasible flow, but this flow may not be optimal.

## 2.7. The Goldberg-Tarjan Maximum Flow Method

Another important algorithm belonging to the dual coordinate ascent class is the maximum flow method of Goldberg and Tarjan ([26] and [27]). This algorithm was developed roughly concurrently with, and entirely independently from, the RELAX family of codes. The original motivation for this algorithm seems to have been quite different than the theory we have developed above; 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 [23]. 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 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 provided 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 could be applied to the case of ε-relaxation.

## 2.8. Complexity Analysis

There are several difficulties in adapting the maximum flow analysis of [26] and [27] to the case of $\varepsilon$-relaxation. The first is in placing a limit on the amount that prices can rise. The approach taken here synthesizes the ideas of [7] with those of [28]-[30]. This methodology can also be applied directly to solving maximum flow problems with arbitrary initial prices.

Another problem is that of *flow looping*. This is discussed in Section 4.4 (see figure 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 this 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 by integer amounts only.

Section 4.4 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 [7], 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 [7], provides an $O(N^2\beta/\varepsilon)$ complexity bound where $\beta$ is a parameter bounded by the maximum simple path length in the network where the length of arc (i, j) is $|a_{ij}|$. 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 section 6 and 7.1). This paper emphasizes *degenerate price rises*, which are critical to overcoming price haggling, and shows that they can be implemented in a way that does not alter the $\varepsilon$-relaxation method's theoretical complexity.

## 2.9. Developments in Scaling

$\varepsilon$-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 *$\varepsilon$-scaling*. In cost scaling algorithms (which have their roots in [23]), one holds $\varepsilon$ fixed and gradually introduces more and more accurate cost data; in $\varepsilon$-scaling, the cost data

are held fixed and $\varepsilon$ 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 $\varepsilon$ may be greater than or equal to d/N. Computational experiments on $\varepsilon$-scaling in the auction algorithm were done in 1979 [8] and again in 1985 [9]. The method of $\varepsilon$-scaling was first *analyzed* in [28], where an algorithm with O(NAlog(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 [29] and [30], 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 $\varepsilon$-relaxation we call *broadbanding,* which will be described later in this paper. Independent discovery of the sweep implementation (there called the wave implementation) is claimed in [30]. 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 $\varepsilon$-relaxation method. Scaling analyses similar to [28] appeared later in such works as [24], [25], 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 $\varepsilon$-scaling. This, in conjunction with the sweep implementation, leads to a simple algorithm with an $O(N^3\log NC)$ complexity. This approach also bypasses the need for the broadbanding modification to the basic form of the $\varepsilon$-relaxation method, introduced in [28]-[30] in conjunction with $\varepsilon$-scaling.

Usually the most challenging part of scaling analysis ([18], [23], [28]-[30], [34], [38], [42]) 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 $\varepsilon$-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 [28] (see also [29], [30]). A key lemma is Lemma 5 of [28], 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 [18] 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 [28], but is also an extension of Corollary 3.1 of [7]. We introduce a measure $\beta(p^0)$ of suboptimality of the initial price vector $p^0$, whereas [28]-[30] use an upper bound on this measure. This extension allows the lemma to be used in contexts other than scaling.

# 3. The ε-Relaxation Method in Detail

To discuss ε-relaxation in detail, we must first be more specific about when the method is actually allowed to change the flow along an arc.

## 3.1. The Admissible Graph

When the ε-relaxation algorithm is performing an up iteration at some node i, it only performs two kinds of flow alterations: flow increases on outgoing $\varepsilon^+$-balanced arcs (i, j) with $f_{ij} < c_{ij}$, and flow decreases on incoming $\varepsilon^-$-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 (multi)graph with node set **N**, an edge (i, j) for each $\varepsilon^+$-balanced arc (i, j) in **A** with $f_{ij} < c_{ij}$, and a reverse edge (j, i) for each $\varepsilon^-$-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 [36], for example), but only contains edges corresponding to arcs that are admissible.

## 3.2. 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 corresponding to edges 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 ascent class of algorithms ([26],[27],[28],[7]) implicitly require push lists or something similar. The first time push lists seem to have been discussed explicitly is in [30], where they are called *edge lists*.

## 3.3. The Exact form 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 $i \in \mathbf{N}$ 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 the 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 (i, j) from the 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

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

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 $\varepsilon$ and stop.)

The serial $\varepsilon$-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 $i \in \mathbf{N}$, in which case it follows that $g_i = 0$ for all $i \in \mathbf{N}$, and that f is feasible.

## 3.4. Basic Lemmas

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

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

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

which may be obtained by combining that the push list is empty and complete with ε-complementary slackness. Therefore, when Step 4 is entered we have

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

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

It follows that step (4) must increase $p_i$, unless $g_i > 0$ 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 $(j, i) \in \mathbf{A}$ becomes $\varepsilon^+$-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 $\varepsilon^-$-balanced as a result of a price rise at i. We have thus shown that all edges added to the admissible graph by step 4 are outgoing from i. **QED.**

**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. **QED.**

We henceforth assume that all arc costs and initial prices are integer multiples of $\varepsilon$. 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 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 statement 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. **QED.**

### 3.5. Finiteness

We now prove that the $\varepsilon$-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 $\varepsilon$-relaxation method terminates with (f, p) satisfying $\varepsilon$-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 $\mathbf{N}^\infty$ 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, $\mathbf{N}^\infty$ is a strict subset of **N**. To preserve $\varepsilon$-CS, we must have after a sufficient number of iterations

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

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

while the sum of surpluses of the nodes in $\mathbf{N}^\infty$ is positive. This means that even with as much flow as arc capacities allow coming out of $\mathbf{N}^\infty$ to nodes $j \notin \mathbf{N}^\infty$, and as little flow as arc capacities allow coming into $\mathbf{N}^\infty$ from nodes $j \notin \mathbf{N}^\infty$, the total surplus $\sum\{g_i \mid i \in \mathbf{N}^\infty\}$ of nodes in $\mathbf{N}^\infty$ 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 $\varepsilon^-$-balanced arc (j, i), or $\varepsilon^+$-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 $\varepsilon^+$-balanced or $\varepsilon^-$-balanced from $\varepsilon^-$-balanced or $\varepsilon^+$-balanced (respectively) an infinite number of times. For this to happen, the price of the adjacent node j must be increased by at least $2\varepsilon$ an infinite number of times. It follows that $p_j \rightarrow \infty$ which contradicts the boundedness of all node prices shown earlier. Therefore the algorithm must terminate. **QED.**

## 3.6. Variations

### 3.6.1. Degenerate Price Rises

Note that when the push list is empty, the price $p_i$ of the current node will be raised at the end of an up iteration even when $g_i = 0$. We call such a price rise *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 very good idea in practice. Similar price changes are very useful in the RELAX family of algorithms.

Following the analysis of directional derivatives and supergradients of section 2.3, one may show that at the end of the iteration, $p_i$ equals $\varepsilon$ plus the *largest* value that maximizes the dual cost with respect to $p_i$ with all other prices kept fixed. An exception is when Step 4 terminates with $g_i = 0$ and the set in (17) empty. In this case, one can show that the dual cost is constant as $p_i$ increases without bound, and there is no largest real value of $p_i$ maximizing the dual cost. We can thus interpret the algorithm as a relaxation method, although "approximate relaxation" may be a better term. If degenerate price rises are omitted, then the up iteration leaves $p_i$ at $\varepsilon$ plus the *smallest* maximizer of the dual objective with the other prices held fixed (refer to figure 6). Except in the above exceptional case, each execution of step 4 corresponds to moving from the neighborhood of one breakpoint of the dual cost to the next .

### 3.6.2. Partial Iterations

Actually, it is not 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. Such algorithms are not constrained to reducing $g_i$ to zero before turning their attention to other nodes. It turns that out that these algorithms retain the finiteness and most of the complexity properties of ε-relaxation, but it might be more appropriate to call them *approximate descent* methods. They become important when one analyzes synchronous parallel implementations of ε-relaxation.

### 3.6.3. Broadbanding

Another useful variation on the basic up iteration, which we call *broadbanding*, is due to Goldberg and Tarjan [28]-[30]. In our terminology, broadbanding amounts to redefining the admissible arcs to be those that are active and have $f_{ij} < c_{ij}$, along 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

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

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

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 by at least ε, which is useful in ε-scaling.

### 3.6.4. 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 ([11], [13], [14]) 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 been discouraging. Although we do not see these results as conclusive, we henceforth assume that the algorithm consists only of up iterations.

# 4. Basic Complexity Analysis

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

## 4.1. The Price Bound β(p)

We now develop the price bound β(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

$$\begin{aligned} d_H(p) &= \max\{0, \textstyle\sum_{(i,j)\in H^+}(p_i - p_j - a_{ij}) - \sum_{(i,j)\in H^-}(p_i - p_j - a_{ij})\} \\ &= \max\{0, p_{s(H)} - p_{t(H)} - \textstyle\sum_{(i,j)\in H^+} a_{ij} + \sum_{(i,j)\in H^-} a_{ij}\} \ . \end{aligned} \tag{22}$$

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 actually attained for some f. The following lemma shows that β(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

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

(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 $\gamma$-CS conditions given by (6a-b) along H and using (22),

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

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. **QED.**

## 4.2. Price Rise Lemmas

We have already established that $\beta(p)$ is a measure of the optimality of p that is intimately connected with $\varepsilon$-complementary slackness. We now show that $\beta(p)$ also places a limit on the amount that prices can rise in the course of the $\varepsilon$-relaxation algorithm. Corollary 3.1 of [7] is adequate for establishing 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 [28], but does not use a general suboptimality measure like $\beta(p)$. The following lemma combines the analysis of [28] with that of [7], 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 Rockafellar's Conformal Realization Theorem, [39], p. 104.)

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

$$p_s^0 - p_t^0 - \sum_{(i,j)\in H^+} a_{ij} + \sum_{(i,j)\in H^-} a_{ij} \le \beta(p^0)\,. \tag{27}$$

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 $p_j \le p_i - a_{ij} + \varepsilon$ for all $(i, j) \in H^+$ and $p_i \le p_j + a_{ij} + \varepsilon$ for all $(i, j) \in H^-$. By adding these conditions along H we obtain

$$-p_s + p_t + \sum_{(i,j)\in H^+} a_{ij} - \sum_{(i,j)\in H^-} a_{ij} \le |H|\,\varepsilon \le (N-1)\varepsilon \ , \qquad (28)$$

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

$$p_t - p_t^0 \le \beta(p^0) + (N-1)\varepsilon \qquad (29)$$

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 increase 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. **QED.**

In some cases, more information can be extracted from $f - f^0$ than in the above proof. For instance, Gabow and Tarjan [25] 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 ascent 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 [25]. This 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 ([11], [13], [43]), which, under certain circumstances typically occurring near the end of execution, occasionally use descent directions corresponding to a more conventional primal-dual method.

### 4.3. Work Breakdown

Now that a limit has been placed on the number of price increases, we must limit the amount of work involved 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 early maximum flow analysis ([26], [27]): 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 $\beta$ 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

$$O( [\sum_{i \in N} d(i)] \ (\beta/\varepsilon + N)) = O(A(\beta/\varepsilon + N)) \ . \qquad (30)$$

**QED.**

**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 $2\varepsilon$ 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. **QED.**

## 4.4. 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 $\varepsilon$-relaxation algorithm.
**Proof:** All "push" operations (executions of steps 2 and 3) can only remove edges from the admissible graph; only price rises can insert edges into the graph. Note also that in lemma 1, we

proved that when edges 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 edges. **QED.**

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 *descendent* 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 [26] and [27], 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 (see section 3.6.2): every up iteration must drive the surplus of its node to zero. Secondly, we assume that the algorithm be 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 [7], 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 edges from the admissible graph, so the 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. **QED.**

**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 cycles is $O(N(\beta/\varepsilon + N))$.

**Proof:** Let $\mathbf{N^+}$ be the set of nodes with positive surplus that have no predecessor with positive surplus, and let $\mathbf{N^0}$ 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 $\mathbf{N^0}$ remain in $\mathbf{N^0}$, and the execution of a complete up iteration at a node $i \in \mathbf{N^+}$ moves i from $\mathbf{N^+}$ to $\mathbf{N^0}$. If no node changed price during a cycle, then all nodes of $\mathbf{N^+}$ will be added to $\mathbf{N^0}$ 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. **QED.**

**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. **QED.**

Figure 8 depicts the sweep implementation.

**Proposition 3.** Under the sweep implementation, if the initial admissable 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 lemmas 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 course of each cycle, which involve 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 $O(N^2(\beta/\varepsilon + N))$. **QED.**

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 $\varepsilon$-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 $\mathbf{N^0}$ 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. [30] 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) $\varepsilon$-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 $\beta$ 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 $\beta^*$ on $\beta$ 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 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 $\beta^*$.

## 5. 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 Figure 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 $\varepsilon$-relaxation algorithm with initial prices and arc flows satisfying $\varepsilon$-complementary slackness, where $\varepsilon = 1/(N+1)$. The initial prices may be arbitrary, so long as there is on 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 $\varepsilon$ and negative values for $a_{ts}$, as long as $\varepsilon = -a_{ts}/(N+1)$ (more generally $\varepsilon = -a_{ts}/(1+$ Largest number of arcs in a cycle containing (t,s))).

Applied like this to the maximum flow problem, $\varepsilon$-relaxation yields an algorithm resembling the maximum flow algorithm of [26]-[27], and having the same complexity. However, it has only one

phase. The first phase of the procedure of [26]-[27] may in hindsight considered to be an application of $\varepsilon$-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 $s_t = +\infty$.

# 6. Scaling Procedures

In general, some sort of *scaling procedure* ([18], [23], [38]) must be used to make the $\varepsilon$-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 $\varepsilon$-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 ([28] and, with Tarjan, [30]), who used $\varepsilon$-scaling. In order to be sure that all price rises are by $\Omega(\varepsilon)$ amounts, both these papers use the broadbanding variant of $\varepsilon$-relaxation as their principal subroutine (though they also present alternatives which are not dual coordinate ascent methods). Here, we will present an alternative cost scaling procedure that results in an overall complexity of $O(N^3 \log NC)$.

## 6.1. 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

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

If the pair $(f', p')$ satisfies 1-complementary slackness (namely $\varepsilon$-complementary slackness with $\varepsilon_2 = 1$) 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), and hence $f'$ 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 $\varepsilon$-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( • ) 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 $(f^t(m), p^t(m))$ 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, . . . , M-1) is

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

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

The starting flow vector $f^0(m+1)$ for the (m+1)st problem may be obtained from $f^t(m)$ 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

$f^0_{ij}(m+1) = f^t_{ij}(m)$ for all balanced arcs (i, j), (37a)

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

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

This procedure implies that the initial admissible graph for the (m+1)st problem has no edges, 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 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

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

In this case, the edge 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.

## 6.2. 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

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

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 $f^t(m)$ 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

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

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

Of course, many variations are possible. For example, it is not necessary to use the sweep implementation to achieve polynomial complexity. Also, it is possible to increase the accuracy of the cost data by factors other than two. Our limited computational experiments on the NETGEN

family of problems seems to indicate that it is more efficient to increase accuracy by a factor between 4 and 8 between consecutive subproblems.

### 6.3. Further Developments in Scaling

Other recent developments in scaling include Gabow and Tarjan's [25], which is also a cost scaling method. Ahuja and Orlin, working jointly with Goldberg and Tarjan, have also developed a *double scaling* method which scales not only $\varepsilon$, but also surpluses and arc capacities, and has complexity $O(NA (\log \log U) \log NC)$, where U is the maximum arc capacity $c_{ij}$ [3] . Furthermore, analysis in [30], drawing on some ideas of Tardos [42], shows that a *strongly polynomial* bound (that is, one polynomial in N and A) may be placed on a properly implemented scaling algorithm.

## 7. The Auction Algorithm

### 7.1. Motivation

Despite the good theoretical complexity bounds available for the scaled form of $\varepsilon$-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 $\varepsilon$ or $2\varepsilon$ per price change. This is the phenomenon of *price haggling*. Essentially, the algorithm is following a "staircase" path in the dual (such as in figure 5), where the individual steps are very small.

Without scaling, the amount of price haggling can be exponential (as in figure 9), so scaling is clearly necessary to make $\varepsilon$-relaxation efficient. However, even with scaling, our computational experiments have shown that 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 cause a dramatic decrease in price haggling, often by orders of magnitude; we contend that they will be necessary in any practical implementation of algorithms of this type.

Even with scaling and degenerate steps, however, we have found $\varepsilon$-relaxation to be several orders of magnitude slower than state-of-the-art sequential codes such as RELAX for large problems. We have not yet experimented with broadbanding and $\varepsilon$-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.

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. We will now develop the theory of this more specialized algorithm.

## 7.2. 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 [9] and [10].

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, of course) 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 is "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 [9] and [10], 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 [9]-[10]. 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 $\gamma$ 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 converge well in practice.

## 7.3. Push Lists and Complexity

If we implement the auction algorithm as a variation of $\varepsilon$-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 actually 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 prices p on the sink nodes, define an *artificial price* $\pi_i$ of each source node i by

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

The reader may confirm that the prices $\pi$, p and the current flow (assignment) f always obey $\gamma$-complementary slackness. The reader may also refer to [10] for a proof that if f is feasible (that is, it is a complete assignment) and (f, $\pi$, p) satisfy $\gamma$-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 $\beta^*$ on the amount that any single $p_j$ can rise. From (a-d) above, all prices are by at least $\gamma$, so there are at most $\beta^*/\gamma$ 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 $\varepsilon$-relaxation, all prices are nondecreasing):

(1) Locate an unassigned source node i.

(2) Examine the elements $(j, p_j')$ of 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 performed in O(d(i)) time, and that between for every two consecutive executions of (4) at a given node i, there must be an increase in the artificial price $\pi_i$ 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 not clear whether the overhead of keeping push lists in the auction algorithm is actually justified in practice. A simpler implementation that has i scan its incident arcs once per bid, whether or not there has been a change in $\pi_i$ , can be shown to have complexity $O(N^3 \log NC)$.

## 7.4. Computational Results

In this section we discuss limited computational experience with a 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. (Note that this check is much less elaborate than the procedure proposed in [30] for the scaled ε-relaxation algorithm. There, a shortest-path type of calculation is used to try and "fit" prices to the current flow. This is an interesting idea which we have not experimented with.) 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.

The version of AUCTION discussed here uses a Gauss-Seidel scheme, 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 [10]. The Gauss-Seidel version is faster, but of course not as amenable to parallel implementation.

Test problems were generated using the 1987 release version of the widely-used public domain generator NETGEN [31]. The AUCTION code was compared with the relaxation FORTRAN code RELAX-II [18], [20]. To give AUCTION truly state-of-the-art competition, we should have used the specially adapted version of RELAX-II, RELAX-IIA. However, experience has shown that RELAX-IIA is only about 15% to 20% faster than RELAX-II. Both codes were run on a MicroVAX II CPU under the VMS 4.6 operating system.

Our results are summarized in figures 11-13. In all runs, the initial value of $\varepsilon$ was nC/2, and $\varepsilon$ was reduced by a factor of 6 for each subproblem. Figure 11 gives the solution times for five problems with equal density (1.5%). Figure 12 gives times for five problems in which the average node degree was 5. Figure 13 shows times for five problems with the same number of nodes, but varying numbers of arcs.

A Jacobi version of the auction algorithm has also been independently implemented and tested on dense assignment problems by Professor J. Kennington and Mr. L. Hatay at Southern Methodist University using a Sequent Balance 21000 computer — a shared memory parallel machine. Figure 14 shows their results with a multiple processor implementation and, the speedup they obtained as a function of the number of CPUs employed. The efficiency of the algorithm for a small number of processors appears quite satisfactory. Note that the speedup with one processor is less than one because, even on a single processor, their parallel code is not as efficient as their serial code.

## 8. Asynchronous Implementation of $\varepsilon$-Relaxation

So far as we know, nobody has been able to show how a theoretical speedup of either the auction or $\varepsilon$-relaxation algorithms may be obtained by a simple synchronous parallel implementation. In this section we will do something quite different: we demonstrate that there is a version of the $\varepsilon$-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 than 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 [15]-[17], [20]. The algorithm given here is more complex than a related algorithm for strictly convex arc costs [15], and requires a novel method of convergence proof.

We now return to the ordinary $\varepsilon$-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

$p_j(i, t)$ : The price of node $j \in F_i \cup B_i$ communicated by j at some earlier time

$f_{ij}(i, t)$ : 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

$g_i(t)$ : The estimate of the surplus of node i at time t given by

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

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

$$p_i(t_0) \geq p_i(j, t_0), \qquad \text{for all } j \in F_i \cup B_i$$
$$f_{ij}(i, t_0) \geq f_{ij}(j, t_0), \qquad \text{for all } j \in F_i \ .$$

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 $t \geq t_0$

$$p_i(t) \geq p_i(j, t''), \qquad \text{for all } j \in F_i \cup B_i, \ t'' \leq t \tag{50}$$

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 $\epsilon$-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 $t \geq t_0$

$$f_{ij}(i, t) \geq f_{ij}(j, t), \qquad \text{for all } j \in F_i, \tag{51}$$

*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 $(i, j) \in \mathbf{A}$, 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

$$\sum_{i \in \mathbf{N}} g_i(t) \leq 0, \qquad \text{for all } t \geq t_0 . \tag{52}$$

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

$$g_i(t) = 0 \qquad \text{for all } i \in \mathbf{N} \tag{53}$$

$$f_{ij}(i, t) = f_{ij}(j, t) \qquad \text{for all } (i, j) \in \mathbf{A} \tag{54}$$

$$p_j(t) = p_j(i, t) \qquad \text{for all } j \in F_i \cup B_i. \tag{55}$$

Termination can be detected by using an adaptation of the protocol for diffusing computations of [22]. 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 $\varepsilon^+$-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 $\varepsilon^-$-balanced. But this is impossible since, when $p_i$ and $p_j$ converge, arc (i, j) cannot become both $\varepsilon^+$-balanced and $\varepsilon^-$-balanced infinitely often.

The second possibility is that there is a nonempty subset of nodes $\mathbf{N}^\infty$ whose prices increase to $\infty$. 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 $\mathbf{N}^\infty$ is a strict subset of **N**. Since the algorithm maintains $\varepsilon$-CS, we have for all sufficiently large t that

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

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

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

## 9. References

[1] Ahuja, R. K., and Orlin, J. B., "A Fast and Simple Algorithm for the Maximum Flow Problem", Working paper, M. I. T., Nov. 1986

[2] Ahuja, R. K., and Orlin, J. B., *An Improved Algorithm for Computing the Min Cycle Mean*, unpublished manuscript, November 1987.

[3] Ahuja, R. K., Orlin, J. B., Goldberg, A. V., and Tarjan, R. E., *Finding Minimum-Cost Flows by Double Scaling*, unpublished manuscript, November 1987.

[4] Awerbuch, B., "Complexity of Network Synchronization", *Journal of the ACM*, Vol. 32, 1985, pp. 804-823

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

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

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

[8] Bertsekas, D. P., "A Distributed Algorithm for the Assignment Problem", Unpublished LIDS Working Paper, M. I. T., March 1979

[9] Bertsekas, D. P., "A Distributed Asynchronous Relaxation Algorithm for the Assignment Problem", *Proc. 24th IEEE Conference on Decision and Control*, Ft Lauderdale, Fla., Dec. 1985, pp. 1703-1704

[10] Bertsekas, D. P., "The Auction Algorithm: A Distributed Relaxation Method for the Assignment Problem", LIDS Report P-1653, M.I.T., March 1987

[11] Bertsekas, D. P., "A Unified Framework for Primal-Dual Methods in Minimum Cost Network Flow Problems", *Math. Programming*, Vol. 32, 1985, pp. 125-145

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

[13] Bertsekas, D. P., and Tseng, P., "Relaxation Methods for Minimum Cost Ordinary and Generalized Network Flow Problems", LIDS Report P-1462, M. I. T., May 1985, to appear in *Operations Research*, January 1988

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

[15] Bertsekas, D. P., and El Baz, D., "Distributed Asynchronous Relaxation Methods for Convex Network Flow Problems", LIDS Report P-1417, M. I. T., Oct. 1984, *SIAM J. on Control and Optimization*, Vol. 25, 1987, pp. 74-85

[16] Bertsekas, D. P., "Distributed Asynchronous Computation of Fixed Points", *Math. Programming*, Vol. 27, 1983, pp. 107-120

[17] Bertsekas, D. P., Tsitsiklis, J. N., and Athans, M., "Convergence Theories of Distributed Asynchronous Computation: A Survey", LIDS Report P-1412, M. I. T., Oct. 1984; also in *Stochastic Programming*, by F. Archetti, G. Di Pillo, and M. Lucertini (eds.), Springer-Verlag, N.Y., 1986, pp. 107-120

[18] Bland, R. G., and Jensen, D. L., "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

[19] Censor, Y. and Lent, A., "An Iterative Row-Action Method for Interval Convex Programming", *J. of Optimization Theory and Applications*, Vol. 34, 1981, pp. 321-352

[20] Chazan, D., and Miranker, W., "Chaotic Relaxation", *Linear Algebra and its Applications*, Vol. 2, 1969, pp. 199-222

[21] Cottle, R. W., and Pang, J. S., "On the Convergence of a block successive over-relaxation method for a class of linear complementarity problems", *Math. Programming Study* 17 , 1982.

[22] Dijkstra, E. W., and Sholten, C.S., "Termination Detection for Diffusing Computations", *Information Processing Letters*, Vol. 11, 1980, pp. 1-4.

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

[24] Gabow, H. N., and Tarjan, R. E., "Faster Scaling Algorithms for Graph Matching", Revised Abstract, 1987.

[25] Gabow, H. N., and Tarjan, R. E., "Faster Scaling Algorithms for Network Problems", July 1987.

[26] Goldberg, A. V., "A New Max-Flow Algorithm", Tech. Mem. MIT/LCS/TM-291, Laboratory for Computer Science, M. I. T., 1985

[27] Goldberg, A. V., and Tarjan, R. E., "A New Approach to the Maximum Flow Problem", Proc. 18th ACM STOC, 1986, pp. 136-146

[28] Goldberg, A. V., "Solving Minimum-Cost Flow Problems by Successive Approximations", extended abstract, submitted to *STOC 87*, Nov 1986

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

[30] Goldberg, A. V., and Tarjan, R. E., "Solving Minimum Cost Flow Problems by Successive Approximation", Proc. 19th ACM STOC, May 1987

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

[32] Luenberger, D. G., *Linear and Nonlinear Programming*, Addison-Wesley, Reading, MA, 1984

[33] Ohuchi, A., and Kaji, I., "Lagrangian dual coordinatewise maximization algorithm for network transportation problems wit quadratic costs", *Networks* 14, 1984

[34] Orlin, J. B., "Genuinely Polynomial Simplex and Non-Simplex Algorithms for the Minimum Cost Flow Problem", Working Paper No. 1615-84, Sloan School of Management, M. I. T., Dec. 1984

[35] Pang, J. S., *On the Convergence of Dual Ascent Methods for Large-Scale Linearly Constrained Optimization Problems*, University of Texas at Dallas, unpublished manuscript, 1984

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

[37] Polyak, B. T., "Minimization of Unsmooth Functions", *USSR Computational Mathematics and Mathematical Physics*, Vol. 9, 1969, pp. 14-29

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

[39] Rockafellar, R. T., *Network Flows and Monotropic Programming*, J. Wiley, N. Y., 1984

[40] Rockafellar, R. T., *Convex Analysis,* Princeton Univ. Press, Princeton, N. J., 1970

[41] Stern, T. E., "A class of decentralized routing algorithms using relaxation", *IEEE Transactions on Communication,* COM-25(10), 1977

[42] Tardos, E., "A Strongly Polynomial Minimum Cost Circulation Algorithm", *Combinatorica*, Vol. 5, 1985, pp. 247-255

[43] Tseng, P., "Relaxation Methods for Monotropic Programming Problems", PhD Thesis, Dept. of Electrical Engineering and Computer Science, M. I. T., May 1986

[44] Zenios, S. A., and Lasken, R. A., "Nonlinear network optimization on a massively parallel connection machine", Report 87-08-03. Decision Sciences Department, The Wharton School, University of Pennsylvania, 1987

[45] Zenios, S. A., and Mulvey, J. M., "Relaxation techniques for strictly convex network problems", *Annals of Operations Research* 5, 1986

## 10. Figures

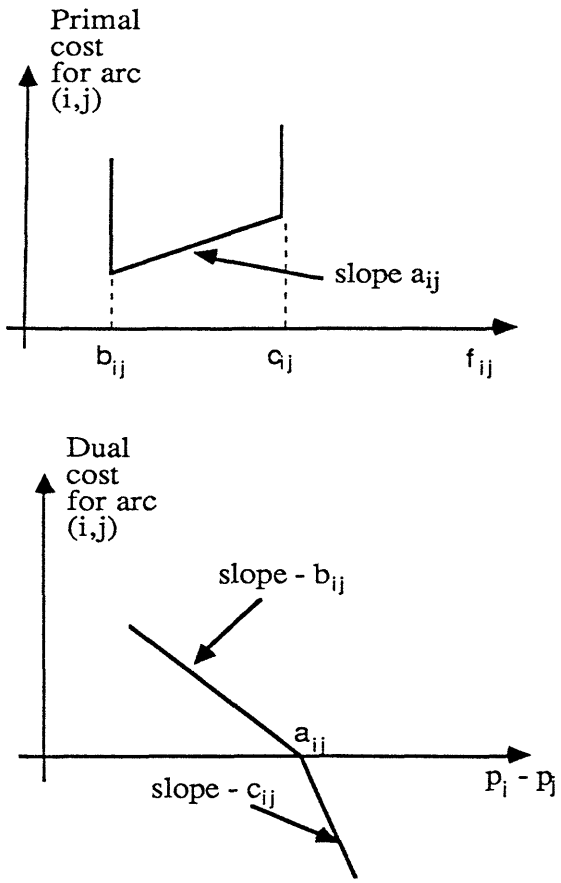



Figure 1: Primal and dual costs for arc (i, j).

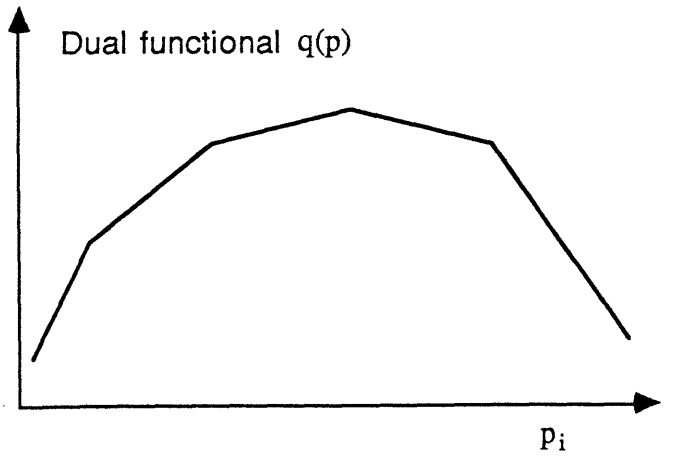



Figure 2: The dual functional $q(p)$ graphed with respect to a single price coordinate.

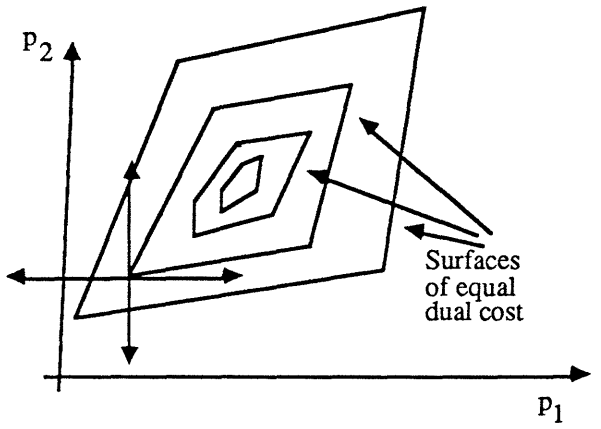



Figure 3: At the indicated point, it is impossible to improve the cost by changing any *single* price.

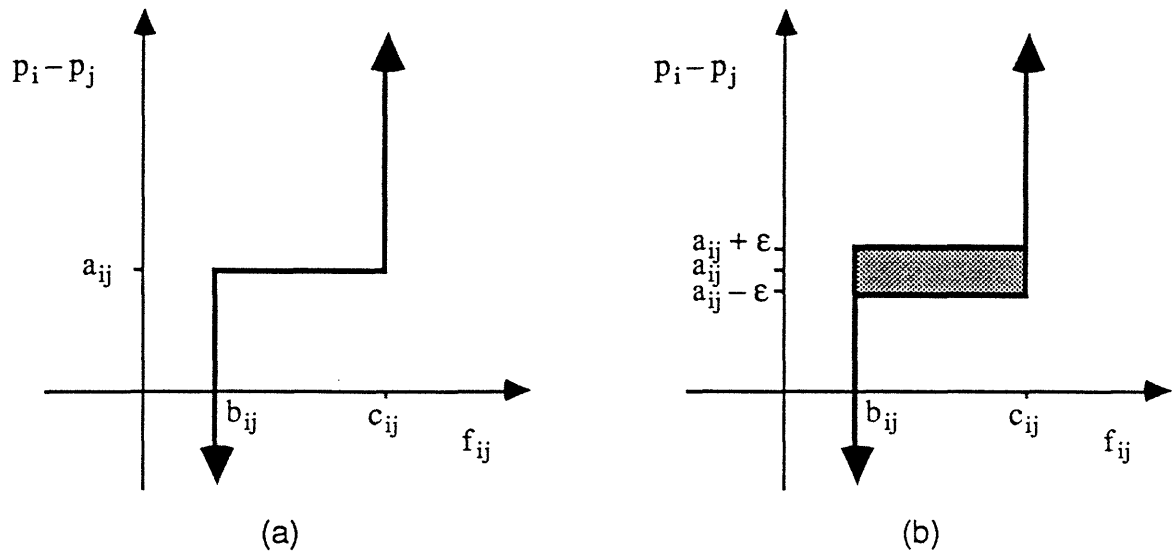



Figure 4: Kilter diagrams for (a) conventional complementary slackness and (b) ε-complementary slackness.

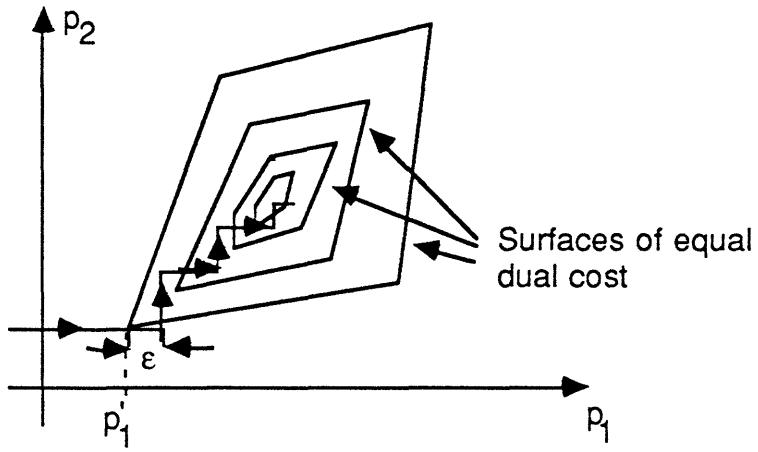



**Figure 5:** When the ith price $p_i$ is chosen for relaxation, it is changed to $p_i' + \varepsilon$, where $p_i'$ is a value of the ith price that maximizes $q(p)$ with all other prices held fixed. When $\varepsilon$ is small, it is possible to approach the optimal solution even if each step does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

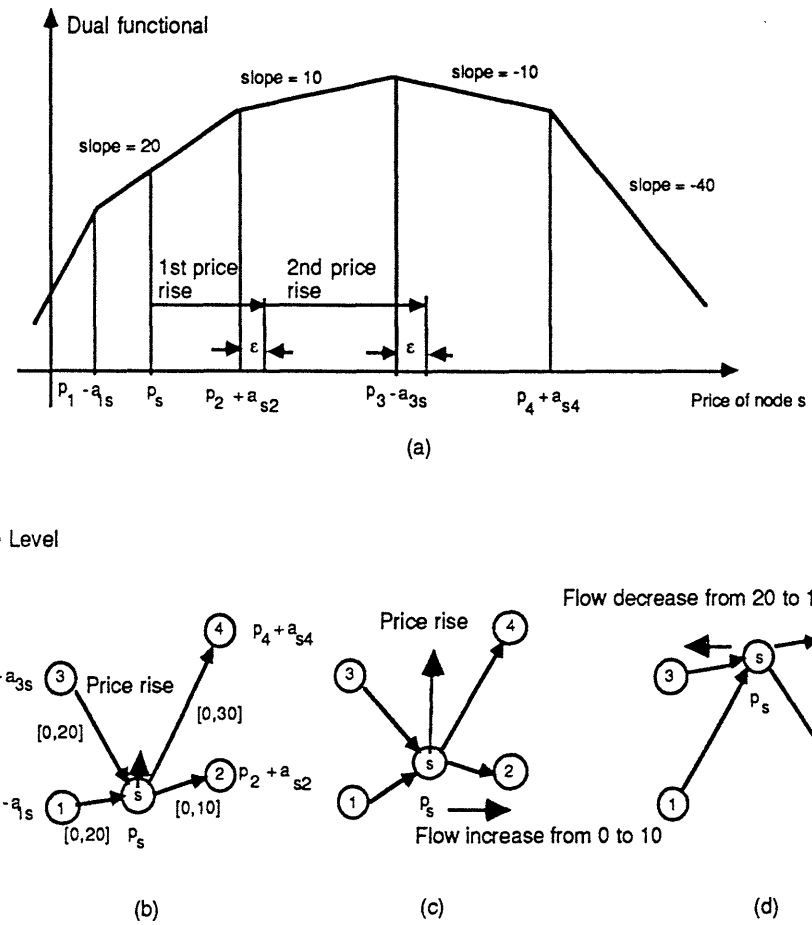



**Figure 6:** Illustration of an up iteration involving a single node s with four incident arcs (1,s), (3,s), (s,2), and (s,4), with feasible arc flow ranges [1,20], [0,20], [0,10], and [0,30], respectively.

**(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 $\varepsilon$ above the breakpoint at which (s,2) becomes balanced (Step 4).

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

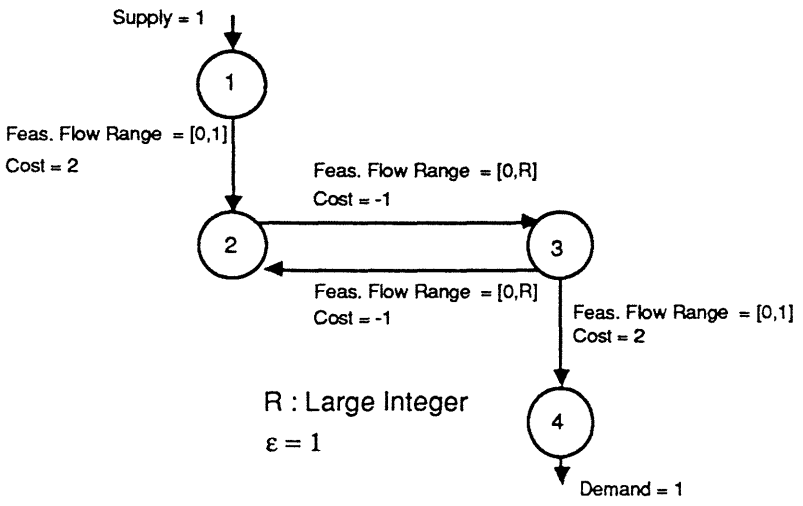



**Figure 7:** Example showing that the importance of keeping the admissible graph acyclic. Initially, we choose f=0, p=0, which do satisfy ε-complementary slackness, but imply a cyclic initial admissible graph. The algorithm will push one unit of flow R times around the cycle 2-3-2, taking $\Omega(R)$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

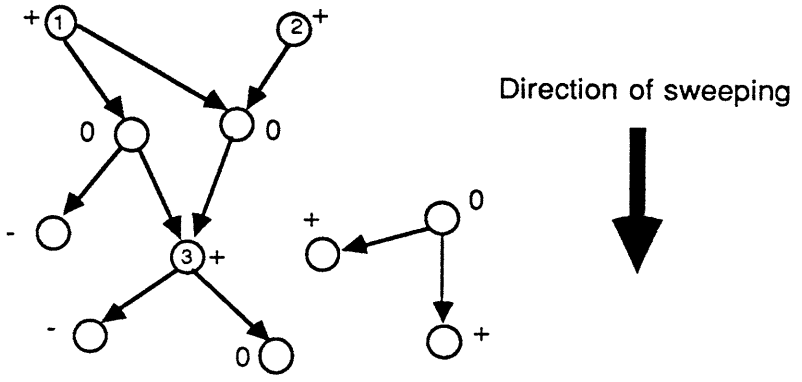
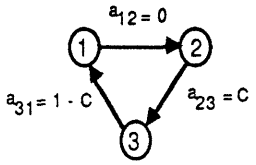
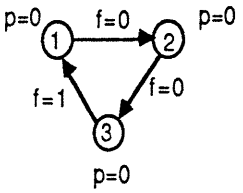



**Figure 8:** Illustration of the admissible graph. A "+" (or "-" or "0") indicates a node with positive (or negative or zero) surplus. The algorithm is operated so that the admissible graph is acyclic at all times. The sweep implementation, based on the linked list data structure, processes high ranking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

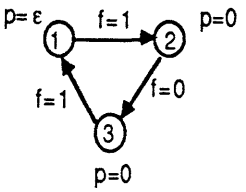



Flow range for arcs:
Arc (1,2): [0,2]
Arc (2,3): [0,1]
Arc (3,1): [0,1]

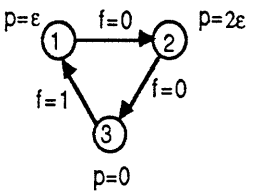
(a) Problem Data



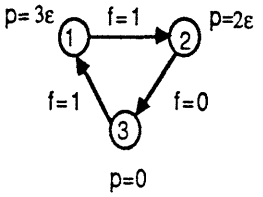

(b) Initial flows and prices




(c) Flows and prices after 1st iteration at node 1




(d) Flows and prices after 2nd iteration at node 2




(e) Flows and prices after 3rd iteration at node 1

**Figure 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+ε and arc (3,1) becomes $\varepsilon^-$-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 $\Omega(C/\varepsilon)$.

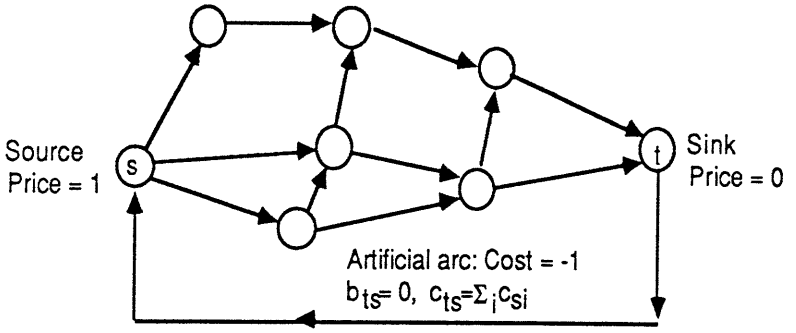



**Figure 10:** Formulation of the max-flow problem.

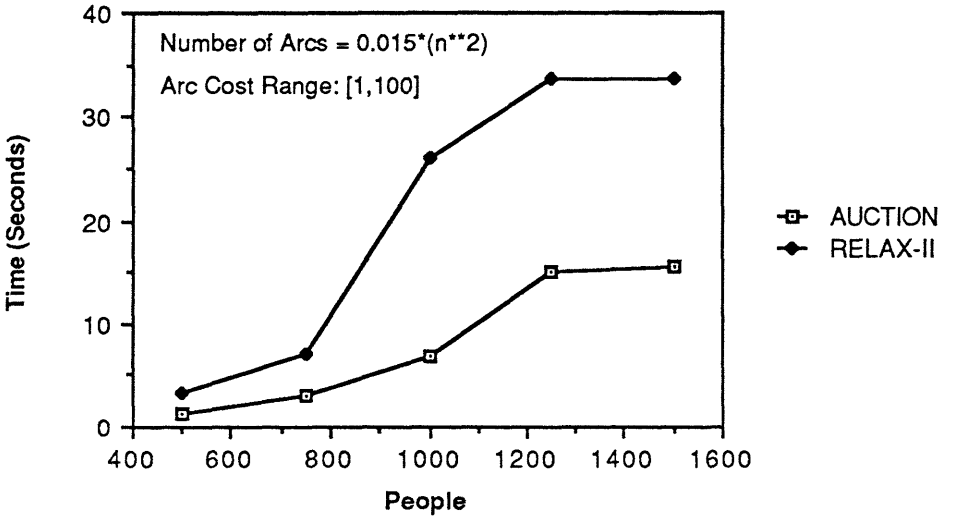



**Figure 11:** Solution times for AUCTION and RELAX-II on a MicroVAX II CPU. All problems generated by NETGEN.

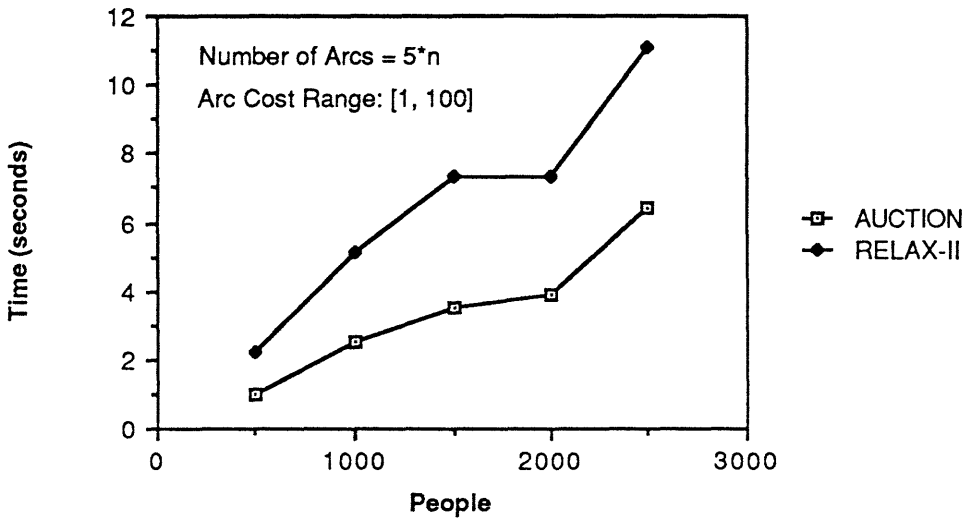



**Figure 12:** Solution times for AUCTION and RELAX-II on a MicroVAX II CPU. All problems produced using NETGEN.

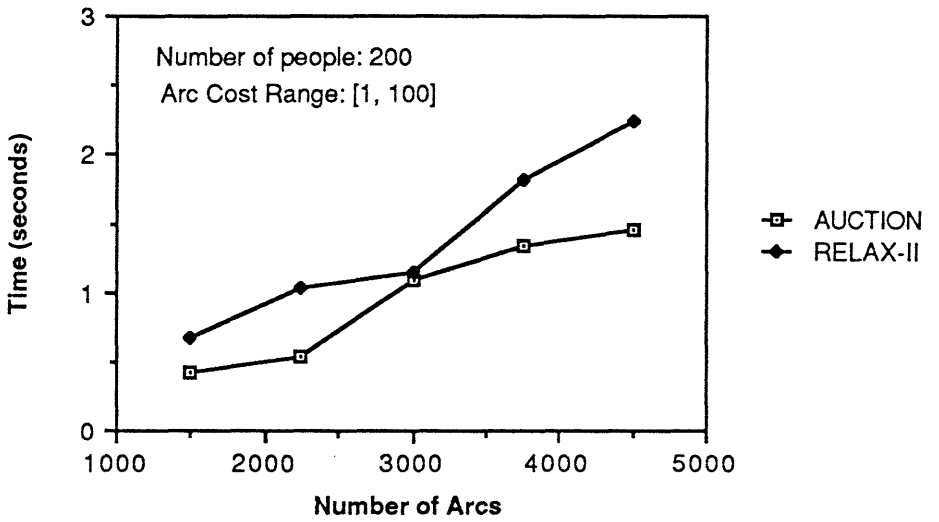



**Figure 13:** Solution times in seconds for AUCTION and RELAX-II on a MicroVAX II CPU, with number of nodes held fixed. All problems created with NETGEN.

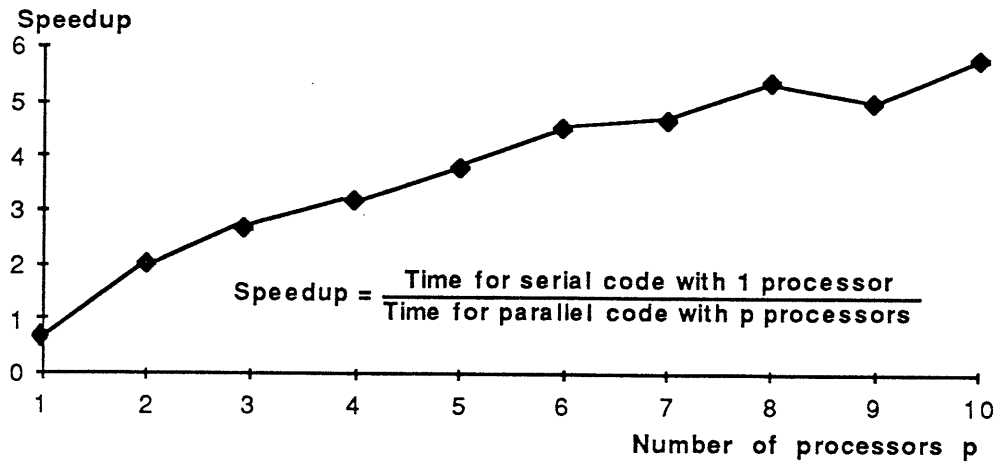



**Figure 14:** Speedup of a parallel implementation of the auction algorithm as a function of the number of processors used in a Sequent Balance 21000 computer. The problem solved is a randomly generated 800x800 fully dense problem with arc cost range 1 - 10000. The time required by the serial auction code using a single processor is 336.13 secs.