

## Distributed Relaxation Methods for Linear Network Flow Problems\*

Dimitri P. Bertsekas

Dept. of Electrical Engineering and Computer Science  
Laboratory for Information and Decision Systems  
Massachusetts Institute of Technology  
Cambridge, Mass. 02139

### Abstract

We consider distributed solution of the classical linear minimum cost network flow problem. We formulate a dual problem which is unconstrained, piecewise linear, and involves a dual variable for each node. We propose a dual algorithm that resembles a Gauss-Seidel relaxation method. At each iteration the dual variable of a single node is changed based on local information from adjacent nodes. In a distributed setting each node can change its variable independently of the variable changes of other nodes. The algorithm is efficient for some classes of problems, notably for the max-flow problem for which it resembles a recent algorithm by Goldberg [11].

### 1. Introduction

Consider a directed graph with set of nodes  $N$  and set of arcs  $A$ . Each arc  $(i,j)$  has associated with it an integer  $a_{ij}$  referred to as the cost coefficient of  $(i,j)$ . We denote by  $f_{ij}$  the flow of the arc  $(i,j)$  and consider the problem

$$\begin{aligned} &\text{minimize } \sum_{(i,j) \in A} a_{ij} f_{ij} && \text{(MCF)} \\ &\text{subject to} \\ &\sum_{(j,i) \in A} f_{ji} - \sum_{(i,j) \in A} f_{ij} = s_i \quad \forall i \in N && \text{(Conservation of flow)} \quad (1) \\ &b_{ij} \leq f_{ij} \leq c_{ij} \quad \forall (i,j) \in A && \text{(Capacity constraints)} \quad (2) \end{aligned}$$

where  $a_{ij}$ ,  $b_{ij}$ ,  $c_{ij}$ , and  $s_i$  are given integers. We assume throughout that there exists at least one feasible solution of this problem. For simplicity we also assume that there is at most one arc associated with each ordered pair of nodes  $(i,j)$ . However this is not an essential restriction and the algorithms and results of the paper can be trivially modified to account for the possibility of multiple arcs joining a pair of nodes.

We first formulate a dual problem associated with (MCF). This problem is the basis for a number of recent works on network flow relaxation methods [2], [3], [5], [6], [7]. We associate a Lagrange multiplier  $p_i$  with the  $i$ th conservation of flow constraint (1). By denoting by  $f$  and  $p$  the vectors with elements  $f_{ij}$ ,  $(i,j) \in A$ , and  $p_i$ ,  $i \in N$  respectively, we can write the corresponding Lagrangian function as

$$\begin{aligned} L(f,p) &= \sum_{(i,j) \in A} a_{ij} f_{ij} \\ &\quad + \sum_i \in N p_i (\sum_{(j,i) \in A} f_{ji} - \sum_{(i,j) \in A} f_{ij} - s_i) \\ &= \sum_{(i,j) \in A} (a_{ij} + p_j - p_i) f_{ij} - \sum_i \in N s_i p_i \quad (3) \end{aligned}$$

The dual function value  $q(p)$  at a vector  $p$  is obtained by minimizing  $L(f,p)$  over all  $f$  satisfying the capacity constraints (3). This leads to the dual problem

$$\begin{aligned} &\text{maximize } q(p) && (4) \\ &\text{subject to no constraint on } p, \\ &\text{with the dual functional } q \text{ given by} \end{aligned}$$

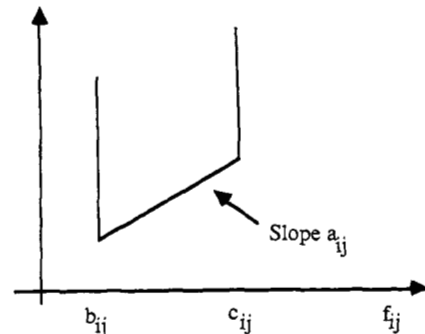
$$\begin{aligned} q(p) &= \min_f \{L(f,p) \mid b_{ij} \leq f_{ij} \leq c_{ij}, (i,j) \in A\} \\ &= \sum_{(i,j) \in A} q_{ij}(p_i - p_j) - \sum_i \in N s_i p_i \quad (5a) \end{aligned}$$

where

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

The function  $q_{ij}$  is shown in Fig. 1. This formulation of the dual problem is consistent with classical duality frameworks [15], [16], but can also be obtained via standard linear programming duality theory [10], [13], [14]. We henceforth refer to (MCF) as the primal problem, and note that, based on standard duality results, the optimal primal cost equals the optimal dual cost. The dual variable  $p_i$  will be referred to as the price of node  $i$ .

Primal cost  $a_{ij} f_{ij}$   
for arc  $(i,j)$



Dual cost  $q_{ij}(p_i - p_j)$   
for arc  $(i,j)$

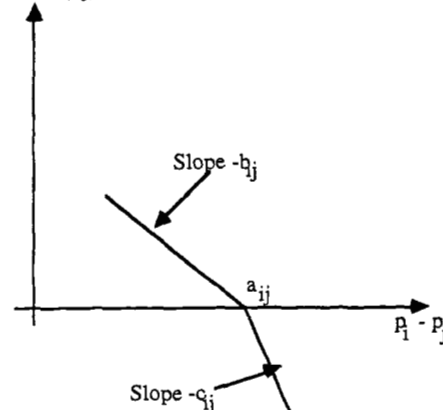


Figure 1: Primal and dual costs for arc  $(i,j)$

The form of the dual cost (5) motivates solution by Gauss-Seidel relaxation (or coordinate descent methods). The idea here is to choose a single node  $i$  and change its price  $p_i$  in a direction of improvement of the dual cost while keeping the other prices unchanged. A key fact [cf. (5)] is that maximization of the dual cost with respect to  $p_i$  requires knowledge of the prices of just the adjacent nodes of  $i$ . As a result the prices of two nonadjacent nodes can be changed in parallel. This leads to distributed methods where the price of each node is adjusted by a separate processor on the basis of price information received from adjacent nodes/processors.

The relaxation idea is straightforward to apply if the arc costs  $a_{ij}f_{ij}$  are strictly convex (rather than linear) functions of the arc flows  $f_{ij}$ . In this case it can be shown that the dual arc cost functions  $q_{ij}$  of (5) are differentiable, so that standard coordinate descent methods from unconstrained nonlinear programming can be applied (see e.g. [13], [18]). It turns out that these methods have remarkable convergence properties explored in [6] and [7]. Indeed these methods work satisfactorily even in a distributed totally asynchronous setting whereby some nodes change their prices faster than others, some nodes communicate their prices faster than others, and there may be arbitrarily large communication delays. The underlying reason for convergence under totally asynchronous conditions is the monotonicity of the algorithmic mapping corresponding to the relaxation process as explained in [7].

The monotonicity property referred to above is also present when the arc costs are linear as assumed in the present paper. Unfortunately there is a fundamental difficulty; the dual cost  $q$  is nondifferentiable (in fact piecewise linear), and the relaxation idea may encounter difficulty at some "corner points" as illustrated in Fig. 2. The problem here is that (in contrast with the case of strictly convex arc costs where the dual functional is differentiable) there are nonoptimal price vectors at which the dual cost cannot be improved by changing any single node price. Two remedies suggest themselves. The first is to allow simultaneous changes of prices of several nodes whenever a single node price change cannot improve the dual cost. Methods of this type have been developed recently, and have proved surprisingly effective, outperforming substantially some of the best primal simplex, and out-of-kilter implementations presently available on standard benchmark problems [3], [5]. The second remedy, suggested in this paper, is to allow single node price changes even if these lead to a deterioration of the dual cost. The idea is illustrated in Fig. 3 which suggests that if the dual cost deterioration due to single node price changes is small, then the algorithm can approach eventually the optimal solution. Indeed we will show that not only this is so, but in fact an exact solution of the problem is obtained in a finite number of iterations owing to the integer nature of the problem data.

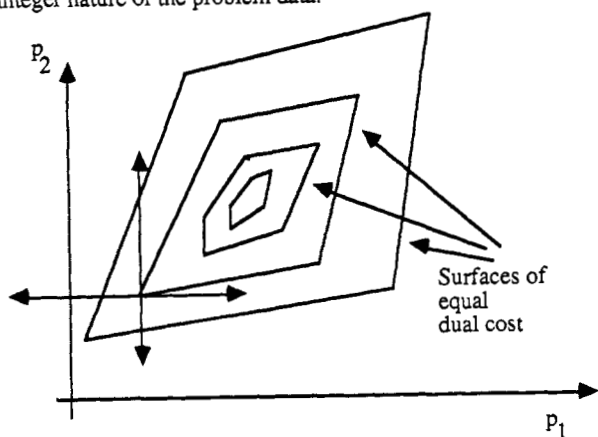


Figure 2: At the indicated point it is impossible to improve the cost by changing any one of the prices.

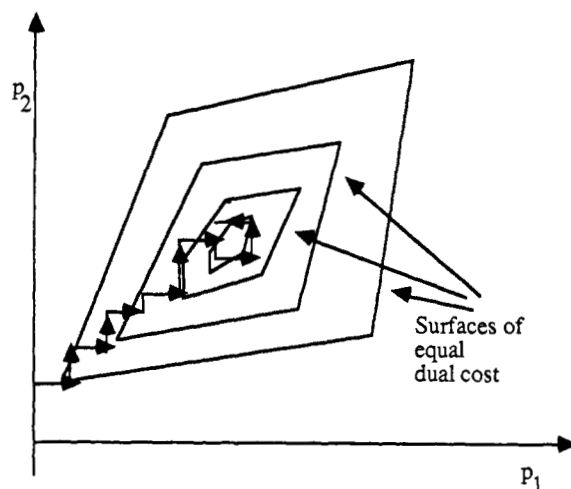


Figure 3: By making small changes in the coordinate directions it is possible to approach the optimal solution even if some steps do not result in a dual cost improvement.

The advantage of the approach of this paper is its suitability for distributed implementation. We envision here the possibility of a separate processor assigned to each node, and changing the node's price on the basis of price and flow information received from adjacent nodes. Similarly as in the case of a strictly convex cost [7], we can show convergence in a distributed asynchronous setting. This is straightforward based on a general framework for showing convergence of distributed asynchronous algorithms [8], [9], but the analysis is not given in the present paper. We do not claim that our algorithm is faster than other methods for the general minimum cost flow problem. The sequential version of the algorithm is, however, competitive with the best sequential methods for some classes of problems as discussed in Section 4. When applied to the max-flow problem the algorithm closely resembles the one developed by Goldberg [11], and refined by Goldberg and Tarjan [12]. The latter algorithm is somewhat more complicated than ours in that it uses two separate phases to obtain an optimal primal solution. Like Goldberg, we show an  $O(N^3)$  worst case complexity bound.

The algorithm of this paper was described for the special case of assignment problems in a 1979 unpublished report by the author. For such problems the algorithm has an interesting interpretation as an auction whereby economic agents compete for resources by making successively higher bids (see [2], [4]). The optimal solution is obtained when the prices of the resources are such that each agent acquires a resource offering maximum profit margin. The algorithm of the present paper can be interpreted similarly as a process of price adjustment that terminates when the supply and demand of flow at each node become equal.

## 2. Optimality Conditions and $\epsilon$ -Complementary Slackness

The necessary and sufficient conditions for a pair  $(f, p)$  to be an optimal primal and dual solution pair are primal feasibility and complementary slackness. To state these conditions we first introduce some terminology.

For any price vector  $p$  we say that an arc  $(i, j)$  is

$$\text{Inactive if } p_i < a_{ij} + p_j \quad (6a)$$

$$\text{Balanced if } p_i = a_{ij} + p_j \quad (6b)$$

$$\text{Active if } p_i > a_{ij} + p_j \quad (6c)$$

For any flow vector  $f$  and node  $i$  the scalar

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

is called the deficit of node  $i$ . It represents the difference of total flow exported and total flow imported by the node.

The primal feasibility and complementary slackness conditions for a vector pair  $(f, p)$  are:

$$d_i = 0, \quad \text{for all } i \in N \quad (8a)$$

$$f_{ij} = b_{ij}, \quad \text{for all inactive arcs } (i,j) \quad (8b)$$

$$b_{ij} \leq f_{ij} \leq c_{ij}, \quad \text{for all balanced arcs } (i,j) \quad (8c)$$

$$f_{ij} = c_{ij}, \quad \text{for all active arcs } (i,j). \quad (8d)$$

We introduce a notion of "approximate" complementary slackness used also in other network relaxation algorithms [5], [6]. For any price vector  $p$  we say that an arc  $(i,j)$  is

$$\epsilon\text{-Inactive} \quad \text{if} \quad p_i < a_{ij} + p_j - \epsilon \quad (9a)$$

$$\epsilon^-\text{-Balanced} \quad \text{if} \quad p_i = a_{ij} + p_j - \epsilon \quad (9b)$$

$$\epsilon\text{-Balanced} \quad \text{if} \quad a_{ij} + p_j - \epsilon \leq p_i \leq a_{ij} + p_j + \epsilon \quad (9c)$$

$$\epsilon^+\text{-Balanced} \quad \text{if} \quad p_i = a_{ij} + p_j + \epsilon \quad (9d)$$

$$\epsilon\text{-Active} \quad \text{if} \quad p_i > a_{ij} + p_j + \epsilon \quad (9e)$$

Given  $\epsilon > 0$  we say that a vector pair  $(f,p)$  satisfies  $\epsilon$ -complementary slackness ( $\epsilon$ -CS for short) if for each arc  $(i,j)$

$$f_{ij} = b_{ij} \quad \text{if} \quad (i,j) \text{ is } \epsilon\text{-inactive} \quad (10a)$$

$$b_{ij} \leq f_{ij} \leq c_{ij} \quad \text{if} \quad (i,j) \text{ is } \epsilon\text{-balanced} \quad (10b)$$

$$f_{ij} = c_{ij} \quad \text{if} \quad (i,j) \text{ is } \epsilon\text{-active} \quad (10c)$$

The algorithm to be described in the next section maintains at all times a price vector  $p$  and an integer flow vector  $f$  satisfying  $\epsilon$ -CS. It terminates when the flow vector  $f$  satisfies the primal feasibility condition  $d_i = 0$  for all  $i \in N$ . A key fact is that if  $\epsilon$  is sufficiently small then the final flow vector  $f$  is optimal. This is shown in the next proposition.

**Proposition 1:** There exists  $\epsilon_1 > 0$  such that if  $\epsilon < \epsilon_1$ , and the integer flow vector  $f$  together with the price vector  $p$  satisfy  $\epsilon$ -CS and primal feasibility ( $d_i = 0$  for all  $i \in N$ ), then  $f$  is optimal for (MCF).

**Proof:** Using the primal feasibility and  $\epsilon$ -CS conditions together with the definition (5) of the dual functional we have

$$\begin{aligned} & \sum_{(i,j)} a_{ij} f_{ij} = \sum_{(i,j)} (a_{ij} + p_j - p_i) f_{ij} - \sum_i s_i p_i \\ & = \sum_{(i,j)} (a_{ij} + p_j - p_i) c_{ij} \\ & \quad \epsilon\text{-active} \\ & + \sum_{(i,j)} (a_{ij} + p_j - p_i) b_{ij} + \sum_{(i,j)} (a_{ij} + p_j - p_i) f_{ij} - \sum_i s_i p_i \\ & \quad \epsilon\text{-inactive} \quad \quad \quad \epsilon\text{-balanced} \\ & = \sum_{(i,j)} (a_{ij} + p_j - p_i) c_{ij} + \sum_{(i,j)} (a_{ij} + p_j - p_i) b_{ij} \\ & \quad \text{active} \quad \quad \quad \text{inactive} \\ & + \sum_{(i,j)} (a_{ij} + p_j - p_i) (f_{ij} - c_{ij}) + \sum_{(i,j)} (a_{ij} + p_j - p_i) (f_{ij} - b_{ij}) - \sum_i s_i p_i \\ & \quad \epsilon\text{-balanced} \quad \quad \quad \epsilon\text{-balanced} \\ & \quad \& \text{ active} \quad \quad \quad \& \text{ inactive} \\ & \leq \sum_{(i,j)} (a_{ij} + p_j - p_i) c_{ij} + \sum_{(i,j)} (a_{ij} + p_j - p_i) b_{ij} + \\ & \quad \text{active} \quad \quad \quad \text{inactive} \\ & + \epsilon \sum_{(i,j)} (c_{ij} - b_{ij}) - \sum_i s_i p_i \\ & \quad \epsilon\text{-balanced} \\ & = q(p) + \epsilon \sum_{(i,j)} (c_{ij} - b_{ij}) \\ & \quad \epsilon\text{-balanced} \\ & \leq q^* + \epsilon \sum_{(i,j)} (c_{ij} - b_{ij}) \quad (11) \end{aligned}$$

where  $q^*$  is the optimal dual cost, which is in turn equal to the optimal primal cost. Take

$$\epsilon_1 = 1 / \sum_{(i,j)} (c_{ij} - b_{ij})$$

Then, for  $\epsilon < \epsilon_1$ ,  $\sum_{(i,j)} a_{ij} f_{ij}$  is strictly within unity of the dual and primal optimal cost. Because  $f$  is integer and the cost coefficients are integer, so is  $\sum_{(i,j)} a_{ij} f_{ij}$ . It follows that  $f$  is optimal.

**Q.E.D.**

The proof of the proposition establishes the bound  $1 / \sum_{(i,j)} (c_{ij} - b_{ij})$  for  $\epsilon$ . For specific classes of problems this bound can be improved as will be seen in Section 4. Note that the price vector  $p$  satisfying the assumptions of Proposition 1 is not optimal. However the proof establishes that  $p$  is within  $O(\epsilon)$  of being optimal, and within close to unity of being optimal when  $\epsilon$  is chosen close to the bound  $1 / \sum_{(i,j)} (c_{ij} - b_{ij})$ .

### 3. The Sequential Version of the Algorithm

The algorithm starts with a fixed value for  $\epsilon > 0$  and with a pair  $(f,p)$  satisfying  $\epsilon$ -CS. The flow vector  $f$  is taken integer initially, and the algorithm preserves this property throughout. At the start of each iteration a node  $i$  with nonzero deficit  $d_i$  is chosen. (If all nodes have zero deficit the algorithm terminates; then  $f$  is primal feasible and together with  $p$  satisfies  $\epsilon$ -CS, so Proposition 1 applies.) At the end of the iteration the deficit  $d_i$  is driven to zero, while another pair  $(f,p)$  satisfying  $\epsilon$ -CS is obtained. During an iteration all node prices stay unchanged except possibly for the price of the chosen node  $i$ . Similarly all arc flows stay unchanged except for the flows of some of the arcs incident to node  $i$ . As a result of these flow changes, the deficit of some of the nodes adjacent to  $i$  is increased or decreased depending on whether the deficit  $d_i$  was positive or negative respectively at the start of the iteration. We describe below the iteration for the two cases where node  $i$  has positive and negative deficit.

#### Negative Deficit Node Iteration (or Up Iteration):

Let  $(f,p)$  satisfy  $\epsilon$ -CS, and let  $i$  be a node with  $d_i < 0$ .

**Step 1:** (Scan adjacent arc) Select a node  $j$  such that  $(i,j)$  is an  $\epsilon^+$ -balanced arc with  $f_{ij} < c_{ij}$  and go to step 2, or select a node  $j$  such that  $(j,i)$  is an  $\epsilon^-$ -balanced arc with  $f_{ji} > b_{ji}$  and go to step 3. If no such node can be found go to step 4.

**Step 2:** (Increase deficit by increasing  $f_{ij}$ ) If  $d_i = 0$  terminate the iteration; else set

$$\begin{aligned} f_{ij} & := f_{ij} + \delta \\ d_i & := d_i + \delta, & d_j & := d_j - \delta \end{aligned}$$

where  $\delta = \min\{-d_i, c_{ij} - f_{ij}\}$ , and go to step 1.

**Step 3:** (Increase deficit by reducing  $f_{ji}$ ) If  $d_i = 0$  terminate the iteration; else set

$$\begin{aligned} f_{ji} & := f_{ji} - \delta \\ d_i & := d_i + \delta, & d_j & := d_j - \delta \end{aligned}$$

where  $\delta = \min\{-d_i, f_{ji} - b_{ji}\}$ , and go to step 1.

**Step 4:** (Increase price of node  $i$ ) Set

$$p_i := \min \{ \{ p_j + a_{ij} + \epsilon \mid (i,j) \in A \text{ and } f_{ij} < c_{ij} \}, \{ p_j - a_{ji} + \epsilon \mid (j,i) \in A \text{ and } b_{ji} < f_{ji} \} \} \quad (12)$$

Go to step 1.

To see that (12) leads to a price increase note that when Step 4 is entered we have  $f_{ij} = c_{ij}$  for all  $(i,j)$  such that  $p_i \geq p_j + a_{ij} + \epsilon$ , and we have  $b_{ji} = f_{ji}$  for all  $(j,i)$  such that  $p_i \geq p_j - a_{ji} + \epsilon$ . Therefore when Step 4 is entered

$$p_i < \min\{ p_j + a_{ij} + \epsilon \mid (i,j) \in A \text{ and } f_{ij} < c_{ij} \}$$

$$p_i < \min\{ p_j - a_{ji} + \epsilon \mid (j,i) \in A \text{ and } b_{ji} < f_{ji} \}$$

It follows that  $p_i$  must be increased via (12). Another issue is that, for Step 4 to be well defined, we must exclude the case where  $f_{ij} = c_{ij}$  for all  $(i,j)$  outgoing from  $i$ , and  $b_{ji} = f_{ji}$  for all  $(j,i)$  incoming to  $i$ , i.e., maximal flow is going out of  $i$  and minimal flow is coming into  $i$ . Since we must have  $d_i \leq 0$  when Step 4 is entered, and by assumption, a feasible solution exists, this case can only happen when  $d_i = 0$ , and furthermore the current flows of arcs incident to  $i$  are

the only feasible. For convenience we assume that this exceptional case does not occur; otherwise we can simply modify step 4 to leave  $p_i$  unchanged and terminate the iteration when this case arises.

**Positive Deficit Node Iteration (or Down Iteration):**

Let  $(f,p)$  satisfy  $\epsilon$ -CS, and let  $i$  be a node with  $d_i > 0$ .

**Step 1:** (Scan adjacent arc) Select a node  $j$  such that  $(i,j)$  is an  $\epsilon^-$ -balanced arc with  $f_{ij} > b_{ij}$  and go to step 2, or select a node  $j$  such

that  $(j,i)$  is an  $\epsilon^+$ -balanced arc with  $f_{ji} < c_{ji}$  and go to step 3. If no such node can be found go to step 4.

**Step 2:** (Reduce deficit by reducing  $f_{ij}$ ) If  $d_i = 0$  terminate the iteration; else set

$$f_{ij} := f_{ij} - \delta$$

$$d_i := d_i - \delta, \quad d_j := d_j + \delta$$

where  $\delta = \min\{d_i, f_{ij} - b_{ij}\}$ , and go to step 1.

**Step 3:** (Reduce deficit by increasing  $f_{ji}$ ) If  $d_i = 0$  terminate the iteration; else set

$$f_{ji} := f_{ji} + \delta$$

$$d_i := d_i - \delta, \quad d_j := d_j + \delta$$

where  $\delta = \min\{d_i, c_{ji} - f_{ji}\}$ , and go to step 1.

**Step 4:** (Reduce price of node  $i$ ) Set

$$p_i := \max \{ \{p_j + a_{ij} - \epsilon \mid (i,j) \in A \text{ and } b_{ij} < f_{ij},$$

$$\{p_j - a_{ji} + \epsilon \mid (j,i) \in A \text{ and } f_{ji} < c_{ji}\} \}$$

Go to step 1.

For step 4 to be well defined we must exclude the case where  $f_{ji} = c_{ji}$  for all  $(j,i)$  incoming to  $i$ , and  $b_{ij} = f_{ij}$  for all  $(i,j)$  outgoing from  $i$ . Similar comments apply as for the earlier case of the up iteration.

The algorithm consists of successively executing up and down iterations until termination. Unfortunately, however, one can construct an example (essentially the same example as the one given in [17], Appendix C) showing that the algorithm may not terminate if up and down iterations are mixed in arbitrary fashion. It is therefore necessary to impose some further assumptions, either on the problem structure or on the method by which up and down iterations are interleaved. The simplest possibility is assumed in the following proposition.

**Proposition 2:** Assume that in the algorithm either only up iterations, or only down iterations are executed. Then the algorithm terminates with  $(f,p)$  satisfying  $\epsilon$ -CS, and with  $f$  being integer and primal feasible.

**Proof:** Assume for concreteness that only up iterations are executed. A similar proof is possible when only down iterations are executed. It should be mentioned here that by (7) we have  $\sum_i d_i = 0$ , so if there is a node with positive deficit, there must also be a node with negative deficit and reversely. It is therefore possible to operate the algorithm using up iterations exclusively.

The following facts can be verified based on the construction of the up iteration:

- 1) The integrality of  $f$  and the  $\epsilon$ -CS property of  $(f,p)$  are preserved throughout the algorithm.
- 2) The prices of all nodes are monotonically nondecreasing during the algorithm.
- 3) Once a node gets nonpositive deficit its deficit stays nonpositive thereafter. (This follows from the fact that an up iteration drives the deficit of the node iterated to zero, and cannot decrease the deficit of its adjacent nodes.)
- 4) If at some time a node has positive deficit it must have never been iterated upon up to that time, and therefore its price must be equal to its initial price. (This a consequence of 3) above and the fact that only nodes with negative deficit are iterated upon by up iterations.)

Based on 2) above there are two possibilities; either a) the prices of a nonempty subset  $N^\infty$  of  $N$  diverge to  $+\infty$ , or else b) the prices of all nodes in  $N$  stay bounded from above.

Suppose that case a) holds. Then, since  $N^\infty$  is nonempty, it follows that the algorithm never terminates implying that at all times there must exist a node with positive deficit which, by 4) above, must have a constant price. It follows that  $N^\infty$  is a strict subset of  $N$ . To preserve  $\epsilon$ -CS, we must have after a sufficient number of iterations

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

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

while the sum of deficits of the nodes in  $N^\infty$  must be negative. This means that even with as much flow as arc capacities allow coming out of  $N^\infty$  to nodes  $j \notin N^\infty$ , and as little flow as arc capacities allow coming into  $N^\infty$  from nodes  $j \notin N^\infty$ , the total deficit  $\sum\{d_i \mid i \in N^\infty\}$  of nodes in  $N^\infty$  is negative. It follows that there is no feasible flow vector contradicting a standing assumption of this paper. Therefore case b) holds (all prices of nodes in  $N$  stay bounded).

We now show by contradiction 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  $\epsilon^-$ -balanced arc  $(j,i)$ , or  $\epsilon^+$ -balanced arc  $(i,j)$  the flow of which 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  $\epsilon^+$ -balanced or  $\epsilon^-$ -balanced from  $\epsilon^-$ -balanced or  $\epsilon^+$ -balanced respectively an infinite number of times. For this to happen, the price of the adjacent node  $j$  must be increased by at least  $2\epsilon$  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. **Q.E.D.**

In other types of relaxation methods for network flow problems [2], [3], [5] - [7], there is an improvement in the dual cost each time there is a price change. This is not true for the price changes effected in Step 4 of the up and the down iterations. However, there is still an interesting interpretation of Step 4 as a dual cost improvement. It can be shown, using the main results of [3], that price changes in Step 4 yield a dual cost improvement of a perturbed problem obtained after the cost coefficients of some of the incident  $\epsilon$ -balanced arcs of the node iterated upon are changed by  $\epsilon$ . In effect, the algorithm improves, with each price change, a slightly perturbed dual cost and ends up with a slightly suboptimal dual solution. The corresponding primal solution, however, is optimal thanks to the rounding introduced by the integer nature of the problem data.

**4. Applications to the Assignment and Max-Flow Problems**

We mentioned earlier that for specific classes of problems one may be able to improve on the estimate for  $\epsilon$  obtained in the proof of Proposition 1. Two such problems are the assignment and max-flow problems.

Consider first the assignment problem. Here the graph is bipartite with  $2N$  nodes, half of which are sources generically denoted by the letter  $i$  ( $s_i = -1$ ), and half of which are sinks generically denoted by the letter  $j$  ( $s_j = 1$ ). The problem is

$$\text{minimize } \sum_i \sum_j a_{ij} f_{ij}$$

$$\text{subject to}$$

$$\sum_j f_{ij} = 1, \quad i = 1, 2, \dots, N$$

$$\sum_i f_{ij} = 1, \quad j = 1, 2, \dots, N$$

$$0 \leq f_{ij} \leq 1, \quad i, j = 1, 2, \dots, N$$

For simplicity we consider the case of a fully dense graph where every source is connected to every sink with an arc. The more general situation can be reduced to the fully dense case by choosing the cost coefficients of the nonexistent arcs sufficiently large. We

assume without loss of generality that  $a_{ij} \geq 0$ ,  $b_{ij} = 0$ ,  $c_{ij} = 1$ , for all  $i$  and  $j$ , and apply the algorithm of the previous section with initial prices and flows equal to zero for all nodes and arcs respectively.

Suppose we use only up iterations. Then, by Proposition 2, the algorithm will terminate in a finite number of iterations with an integer flow vector. All arc flows will be 0 or 1 with only one arc with unity flow incident to any one node. Bearing this in mind we can strengthen the calculations of the proof of Proposition 1. Since only  $N$  of the flows  $f_{ij}$  are nonzero, the summations over all  $(i,j)$  in (11) can be reduced to summations over  $i$  yielding the bound

$$\sum_i \sum_j a_{ij} f_{ij} \leq q^* + \epsilon N$$

It is therefore sufficient to take  $\epsilon < 1/N$  to guarantee optimality of the finally obtained assignment.

We now assume that  $\epsilon = 1/(N-1)$  and estimate the worst case complexity of the algorithm. We first observe that the finally obtained prices of all nodes cannot exceed  $\max\{a_{ij} \mid i,j = 1, \dots, N\} + 2\epsilon$ . To see this note that when the algorithm terminates, the last sink to be assigned to a source will have positive deficit up to the time of termination, and therefore at termination will still have price 0 (fact 4 in the the proof of Proposition 2). This, together with the  $\epsilon$ -CS requirement imply the upper bound  $\max\{a_{ij} \mid i,j = 1, \dots, N\} + \epsilon$  for the source prices. Adding  $\epsilon$  to this we obtain an upper bound on the sink prices in view of  $\epsilon$ -CS. Since  $\epsilon = 1/(N-1)$ , prices increase in increments which are multiples of  $\epsilon$ . We therefore obtain an upper bound of  $O(N^2 \max\{a_{ij} \mid i,j = 1, \dots, N\})$  for the total number of node price changes. This bound can be shown by example to be tight [1], and is somewhat disappointing in that it implies that the algorithm is not polynomial. However when  $\max\{a_{ij} \mid i,j = 1, \dots, N\}$  is relatively small the algorithm may be competitive with other algorithms, particularly when its potential for distributed implementation is considered. For the case of the bipartite matching problem where  $a_{ij}$  is either 0 or 1, it can be seen using the preceding arguments that the computational complexity of the sequential version of the algorithm is  $O(N^3)$ .

Consider next the max-flow problem. We adopt here a formulation shown in Fig. 4. In particular all arcs have 0 cost coefficient, except for the artificial arc  $(t,s)$  connecting the sink  $t$  with the source  $s$  which has cost coefficient  $-1$ . We assume that  $s_i = 0$  for all  $i$ , and  $b_{ij} = 0 < c_{ij}$  for all arcs  $(i,j)$  other than  $(t,s)$ . The flow bounds  $b_{ts}$  and  $c_{ts}$  are taken so that  $b_{ts} < 0$  and  $\sum_i c_{si} < c_{ts}$ . We apply the algorithm using up iterations only. The initial prices and arc flows are 0 except for the following: The price of the source  $s$  is taken to be  $1 + \epsilon$ , the flows of all outgoing arcs  $(s,i)$  from  $s$  equal the corresponding upper bound  $c_{si}$ , and the flow of the artificial arc  $(t,s)$  equals  $\sum_i c_{si}$ .

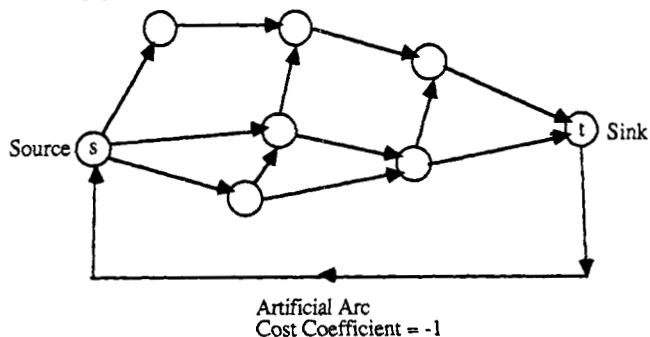


Figure 4: Formulation of the max-flow problem. All arcs have cost coefficient 0 except for the artificial arc that has cost coefficient  $-1$ . We assume that  $b_{ij} = 0 < c_{ij}$ , for all arcs  $(i,j)$  other than  $(t,s)$ , and we take  $b_{ts} < 0$  and  $c_{ts} > \sum_i c_{si}$ .

This choice of initial conditions implies that all nodes have initially zero or negative deficit, except for the sink node  $t$  which has deficit equal to  $\sum_i c_{si}$ . Since at least one node must have positive

deficit prior to termination, and all nodes with nonpositive initial deficit will have nonpositive deficit throughout the algorithm, it follows that the sink node  $t$  will have positive deficit at all times prior to termination, and that the algorithm will terminate when the deficit of  $t$  will become 0. As a result the price of  $t$  will stay constant at 0.

Because the arc  $(t,s)$  is initially  $\epsilon$ -balanced, and  $b_{ts} < 0$ , any price increase of  $p_s$  will make  $d_s$  positive which is not allowed by the algorithm. It follows that throughout the algorithm the price of the source node  $s$  will stay constant at  $1 + \epsilon$ .

We claim that if  $\epsilon < 1/(N-2)$ , where  $N$  is the number of nodes, then the algorithm terminates with an optimal flow vector. To see this consider the flow vector obtained at termination. If it were not optimal, there would exist an augmenting path  $(s, i, j, \dots, k, t)$  from  $s$  to  $t$  where each arc on the path would have flow less than the upper bound, or more than the lower bound depending on whether it is forward or backward oriented. In view of the  $\epsilon$ -CS requirement we must have at termination

$$1 + \epsilon = p_s$$

$$p_s - \epsilon \leq p_i$$

$$p_i - \epsilon \leq p_j$$

$$\dots$$

$$p_t - \epsilon \leq p_k$$

Since  $p_t = 0$ , and the number of arcs on the augmenting path can be at most  $N - 1$ , we obtain by adding the inequalities above

$$1 \leq (N-2)\epsilon$$

Therefore, if  $\epsilon$  is chosen to be less than  $1/(N-2)$ , there can be no augmenting path at termination, and the flow vector obtained must be optimal.

We now assume that  $\epsilon = 1/(N - 1)$ , and estimate the worst case complexity of the algorithm. We first note that at termination we have  $0 \leq p_i \leq 2$  for all nodes  $i$ . This follows from the fact that initially

every node is connected to either  $s$  or  $t$  with a path of  $\epsilon$ -balanced arcs, and it is seen that this property is preserved at each iteration of the algorithm. Since all price changes will be in increments of multiples of  $\epsilon$  (in view of  $\epsilon = 1/(N - 1)$ ), it follows that the price of each node will be increased at most  $O(N)$  times during the algorithm.

The dominant computational requirements of the algorithm can be divided in three parts:

- 1) The computation required for price increases in Step 4 of the up iterations.
- 2) The computation required for Steps 2 or 3 for which the flow of the corresponding arc is set to its upper or its lower bound.
- 3) The computation required for Steps 2 or 3 for which the flow of the corresponding arc is set to a value strictly between its upper and its lower bound.

Since there are  $O(N)$  price increases for each node, the requirements in 1) above are  $O(NA)$  operations, where  $A$  is the number of arcs.

Whenever an arc flow is set to either the upper or the lower bound due to an iteration at one of the end nodes, it takes a price increase of at least  $2\epsilon$  by the opposite end node before the arc flow can change again. Therefore there are  $O(N)$  Steps 2 or 3 per arc for which the flow of the arc is set to its upper or lower bound, and the total requirements for 2) above are  $O(NA)$  operations.

There remains to estimate the computational requirements for 3) above. For this we introduce an order for choosing nodes in iterations. A cycle is a set of iterations whereby all nodes are chosen once in a given order, and an up iteration is executed at each node having negative deficit at the time its turn comes. We henceforth assume that the algorithm is operated in cycles, and proceed to show that the number of cycles up to termination is  $O(N^2)$ . Let  $M = \max\{p_i \mid d_i < 0\}$ , and consider the effect on  $M$  of a single cycle of iterations. There are three possibilities:

- a)  $M$  increases during the cycle. Then the price of some node must increase during the cycle. To see this note that if all node prices were to stay constant during the cycle and  $M$  were to increase, the deficit of some node  $i$  with  $p_i > M$  and  $d_i = 0$  at the start of the cycle must become negative during the cycle - this cannot happen because

all nodes with negative deficit at the start of the cycle have price lower than  $p_i$ , and a node can change its deficit from zero to negative only through an iteration at an adjacent node with higher price. Therefore there is at least one node price that will increase in each cycle for which  $M$  increases. Since the number of price increases per node is  $O(N)$ , it follows that the number of cycles during which  $M$  increases is  $O(N^2)$ . Furthermore the sum of increases in  $M$  is bounded above by the sum of price increases of all nodes which is less than  $2(N-1)$ , since  $0 \leq p_i \leq 2$  for all  $i$  and  $p_s = 1 + \epsilon$ ,  $p_t = 0$ .

b)  $M$  decreases during the cycle. Since  $M \geq 0$  the sum of decreases in  $M$  can exceed the sum of increases in  $M$  by no more than the maximum price value which was shown to be no more than 2. Therefore the sum of decreases in  $M$  is less than  $2N$ . Since  $M$  can decrease only in multiples of  $\epsilon = 1/(N-1)$ , we obtain that the number of cycles during which  $M$  decreases is  $O(N^2)$ .

c)  $M$  stays the same during the cycle. Again we claim that the price of some node must increase during the cycle because the deficit of each node  $i$  with  $d_i \leq 0$  and  $p_i = M$  at the beginning of the cycle will be zero at the end of the cycle. To see this note that the deficit of such a node is either zero or is set to zero when its iteration is performed, and can only decrease through an iteration at some adjacent node with negative deficit and higher price. If all prices stay constant during the cycle such an adjacent node does not exist. Therefore some node price must increase during the cycle, and, by the argument given in a) above, the number of cycles during which  $M$  stays the same is also  $O(N^2)$ .

Thus we have shown that the total number of cycles performed by the algorithm is  $O(N^2)$ . For each cycle there can be only one arc flow per node set to a value strictly between the upper and lower arc flow bound in Step 2 or 3. Therefore the total number of operations required for these steps [cf. 3) above] is  $O(N^3)$ . Adding the computational requirements for 1) and 2) calculated earlier we obtain an  $O(N^3) + O(NA)$  or  $O(N^3)$  worst case complexity bound for the version of the algorithm that is operated in cycles.

## References

- [1] Bertsekas, D. P., "A Distributed Algorithm for the Assignment Problem", Unpublished LIDS Report, M. I. T., March 1979
- [2] Bertsekas, D. P., "A New Algorithm for the Assignment Problem", Math. Progr., Vol. 21, 1981, pp. 152-171
- [3] Bertsekas, D. P., "A Unified Framework for Primal-Dual Methods in Minimum Cost Network Flow Problems", Math. Progr., Vol. 32, 1985, pp. 125-145
- [4] 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.
- [5] 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 Journal
- [6] Bertsekas, D. P., Hossein, P., and Tseng, P., "Relaxation Methods for Network Flow Problems with Convex Arc Costs", LIDS Report P-1523, Dec. 1985, to appear in SIAM J. on Control and Optimization
- [7] 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, to appear in SIAM J. on Control and Optimization, 1986
- [8] Bertsekas, D. P., "Distributed Asynchronous Computation of Fixed Points", Math. Programming, Vol. 27, 1983, pp. 107-120
- [9] 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. 107-139
- [10] Dantzig, G. B., Linear Programming and Extensions, Princeton Univ. Press, Princeton, N.J., 1963
- [11] Goldberg, A. V., "A New Max-Flow Algorithm", Tech. Mem. MIT/LCS/TM-291, Laboratory for Computer Science, M. I. T., 1985

[12] Goldberg, A. V., and Tarjan, R. E., "A New Approach to the Maximum Flow Problem", STOC 86, 1986

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

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

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

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

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

[18] Zangwill, W., Nonlinear Programming: A Unified Approach, Prentice-Hall, Englewood Cliffs, N. J., 1969

\* Supported by Grant NSF - ECS - 8217668