September 1990 LIDS - P - 1998

PARALLEL ASYNCHRONOUS PRIMAL-DUAL METHODS FOR THE MINIMUM COST FLOW PROBLEM[1]

by

Dimitri P. Bertsekas[2] and David A. Castañon[3]

Abstract

In this paper we discuss the parallel asynchronous implementation of the classical primal-dual method for solving the linear minimum cost network flow problem. Multiple augmentations and price rises are simultaneously attempted starting from several nodes and using possibly outdated price and flow information. The results are then merged asynchronously subject to rather weak compatibility conditions. We show that this algorithm is valid, terminating finitely to an optimal solution. We also present computational results using an Encore Multimax that illustrate the speedup that can be obtained by parallel implementation.

[1] This work was supported in part by the BM/C3 Technology branch of the United States Army Strategic Defense Command.

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

[3] ALPHATECH, Inc., Burlington, Mass., 01803.

1. INTRODUCTION

Consider a directed graph with node set $\mathcal{N}$ and arc set $\mathcal{A}$, with each arc (i,j) having a cost coefficient a_{ij}. Let f_{ij} be the flow of the arc (i,j). The minimum cost flow (or transshipment) problem is

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

subject to

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

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

where a_{ij}, b_{ij}, c_{ij}, and s_i are given integers. We assume that there exists at most one arc in each direction between any pair of nodes, but this assumption is made for notational convenience and can be easily dispensed with.

An important method for solving this problem is the primal-dual method due to Ford and Fulkerson [FoF57], [FoF62]. The basic idea is to maintain a price for each node and a flow for each arc which satisfy complementary slackness. The method makes progress towards primal feasibility by successive augmentations along paths with certain properties and by making price changes to facilitate the creation of paths with such properties (see the description in the next section). The paths and the corresponding price adjustments can also be obtained by a shortest path computation (see the next section). There are several variations of the method, known by a variety of names. For example, the recent survey paper by Ahuja, Magnanti, and Orlin [AMO89] calls "successive shortest path method" the variation whereby the search for the augmenting path is initiated from a single node with positive surplus; this is the version of primary interest in this paper and will be referred to as "primal-dual method". In another variation, called "primal-dual method" in [AMO89], the search for augmenting paths is initiated simultaneously from all nodes with positive surplus. This method is also closely related to an algorithm of Busaker and Gowen [BuG61], which also involves augmentations along certain shortest paths. We will not be concerned with this second variation of the primal-dual method since it does not lend itself to the kind of parallelization that we consider.

The classical version of the primal-dual method is serial in nature; only one augmenting path is computed at a time. In a recent paper, Balas, Miller, Pekny, and Toth [BMP89] introduced an interesting new idea for parallelization of the primal-dual method as applied to the assignment problem. In particular, they proposed the parallel construction of several shortest augmenting paths, starting from different nodes. They have shown that if these paths are pairwise disjoint, they can all

be used to modify the current flow; to preserve complementary slackness, the node prices should be raised to the maximum of the levels that would result from each individual shortest path calculation. In [BeC90], we have shown the validity of an asynchronous parallel implementation of the Hungarian method, which is an extension of the synchronous parallel Hungarian method of Balas et al.

In this paper we generalize the synchronous assignment algorithm of Balas et al [BMP89] and our asynchronous version [BeC90] to the general network flow problem (LNF). We show that the corresponding primal-dual method converges finitely to an optimal solution when implemented asynchronously, guided by our earlier analysis of [BeC90]. Based on computational experiments with assignment problems on the Encore Multimax shared memory multiprocessor [BeC90], the asynchronous implementation is often faster than its synchronous counterpart. We discuss combinations of the primal-dual method with single node relaxation (coordinate ascent) iterations, and we similarly show that the combined algorithms work correctly in a parallel asynchronous context.

In the next section we describe synchronous and asynchronous parallel versions of the primal-dual algorithm. In Section 3 we prove the validity of these algorithms, showing that they terminate with an optimal solution in a finite number of steps, assuming that the problem is feasible. The primal-dual method can be substantially accelerated by combining it with single node relaxation iterations of the type introduced in [Ber82]. In Section 4 we show how such combinations can be implemented in a parallel asynchronous setting. Finally, in Section 5 we briefly discuss a synchronous implementation and we report on the results of our computational tests.

2. THE PARALLEL ASYNCHRONOUS PRIMAL-DUAL METHOD

We introduce some terminology and notation. We denote by f the vector with elements f_{ij}, $(i,j) \in \mathcal{A}$. We refer to b_{ij} and c_{ij}, and the interval $[b_{ij}, c_{ij}]$ as the *flow bounds* and the *feasible flow range* of arc (i,j), respectively. We refer to s_i as the *supply* of node i. We refer to the contraints (1) and (2) as the *conservation of flow constraints* and the *capacity constraints* respectively. A flow vector satisfying both of these constraints is called *feasible*, and if it satisfies just the capacity constraints, it is called *capacity-feasible*. If there exists at least one feasible flow vector, problem (LNF) is called *feasible* and otherwise it is called *infeasible*. For a given flow vector f, we define the *surplus* of node i by

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

We introduce a dual variable p_i for each node i, also referred to as the *price of node i*. A flow-

price vector pair (f,p) is said to satisfy the *complementary slackness* conditions (CS for short) if f is capacity-feasible and

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

$$b_{ij} < f_{ij} \quad \Rightarrow \quad p_i \geq a_{ij} + p_j \qquad \forall\, (i,j) \in \mathcal{A}. \tag{4b}$$

For a pair (f,p), feasibility of f and CS are the necessary and sufficient conditions for f to be optimal and p to be an optimal solution of a certain dual problem (see eg. [Roc84] or [BeT89]).

The primal-dual method maintains a pair (f,p) satisfying CS, such that f is capacity-feasible. The method makes progress towards optimality by reducing the total absolute surplus $\sum_{i\in\mathcal{N}} |g_i|$ by an integer amount at each iteration, as we now describe.

For a given capacity-feasible f, an *unblocked path* P is a path $(i_1, i_2, \ldots, i_k)$ such that for each $m = 1, \ldots, k-1$, either (i_m, i_{m+1}) is an arc with $f_{i_m i_{m+1}} < c_{i_m i_{m+1}}$ (called a *forward* arc) or (i_{m+1}, i_m) is an arc with $b_{i_{m+1} i_m} < f_{i_{m+1} i_m}$ (called a *backward* arc). We denote by P^+ and P^- the sets of forward and backward arcs of P, respectively. The unblocked path P is said to be an *augmenting path* if

$$g_{i_1} > 0, \qquad g_{i_k} < 0.$$

An *augmentation* along an augmenting path P consists of increasing the flow of the arcs in P^+ and decreasing the flow of the arcs in P^- by the common positive increment δ given by

$$\delta = \min \; g_{i_1}, -g_{i_k}, \{c_{mn} - f_{mn} \mid (m,n) \in P^+\}, \{f_{mn} - b_{mn} \mid (m,n) \in P^-\} \quad . \tag{5}$$

Given a price vector p, the *reduced cost* of arc (i,j) is given by

$$r_{ij} = a_{ij} + p_j - p_i. \tag{6}$$

If (f,p) is a pair satisfying the CS condition (4) and P is an unblocked path with respect to f, the *cost length* of P is defined by

$$C(P) = \sum_{(i,j)\in P^+} a_{ij} - \sum_{(i,j)\in P^-} a_{ij} \tag{7}$$

and the *reduced cost length* of P is defined by

$$R(p,P) = \sum_{(i,j)\in P^+} r_{ij} - \sum_{(i,j)\in P^-} r_{ij}. \tag{8}$$

Note that by CS, we have $r_{ij} \geq 0$ for all $(i,j) \in P^+$ and $r_{ij} \leq 0$ for all $(i,j) \in P^-$, so $R(p,P) \geq 0$. For a pair of nodes i and j, let $\mathcal{P}_{ij}(f)$ be the set of unblocked paths starting at i and ending at j, and let

$$v_{ij}(f,p) = \begin{cases} \min_{P\in\mathcal{P}_{ij}(f)} R(p,P) & \text{if } \mathcal{P}_{ij}(f) \text{ is nonempty} \\ \infty & \text{otherwise.} \end{cases} \tag{9}$$

If there exists at least one node j with $g_j < 0$, the *distance* of i is defined by

$$d_i = \begin{cases} \min_{\{j|g_j<0\}} v_{ij}(f,p) & \text{if } g_i \geq 0 \\ 0 & \text{otherwise.} \end{cases} \tag{10}$$

The typical primal-dual iteration starts with a pair (f,p) satisfying CS and generates another pair $(\bar{f},\bar{p})$ satsfying CS as follows:

Primal-Dual Iteration:

Choose a node i with $g_i > 0$. (If no such node can be found, the algorithm terminates. There are then two possibilities: (1) $g_i = 0$ for all i, in which case f is optimal since it is feasible and satisfies CS together with p; (2) $g_i < 0$ for some i, in which case problem (LNF) is infeasible.) Let $\bar{j}$ and $\bar{P}$ be the minimizing node with $g_j < 0$ and corresponding augmenting path in the definition of the distance d_i (cf. Eqs. (9), (10)), that is,

$$\bar{j} = \arg \min_{\{j|g_j<0\}} v_{ij}(f,p), \tag{11}$$

$$\bar{P} = \arg \min_{P \in \mathcal{P}_{ij}(f)} R(p,P). \tag{12}$$

Change the node prices according to

$$\bar{p}_j = p_j + \max\{0, d_i - v_{ij}(f,p)\}, \qquad \forall\, j \in \mathcal{N}, \tag{13}$$

and perform an augmentation along the path $\bar{P}$, obtaining a new flow vector $\bar{f}$.

We note that the primal-dual iteration can be executed by a shortest path computation. To see this, consider the *residual graph*, obtained from the original by assigning length r_{ij} to each arc (i,j) with $f_{ij} < c_{ij}$, by replacing each arc (i,j) with $f_{ij} = c_{ij}$ by an arc (j,i) with length $-r_{ij}$, and by replacing each arc (i,j) with $b_{ij} < f_{ij} < c_{ij}$ with two arcs (i,j) and (j,i) with length zero [the reduced cost of (i,j), cf. the CS condition (4)]. Then the augmenting path $\bar{P}$ is a shortest path in the residual graph, over all paths starting at the node i and ending at a node j with $g_j < 0$. Note that by the CS condition, all arc lengths are nonnegative in the residual graph, so Dijkstra's method can be used for the shortest path computation.

The results of the following proposition are well known (see e.g. [AMO89], [Law76], [PaS82], [Roc84]) and will be used in what follows:

Proposition 1: If problem (LNF) is feasible, then a node $\bar{j}$ and an augmenting path $\bar{P}$ satisfying Eqs. (11) and (12) exist. Furthermore, if $(\bar{f},\bar{p})$ is a pair obtained by executing a primal-dual iteration on a pair (f,p) satisfying CS, the following hold:

(a) If f consists of integer flows, the same is true for $\bar{f}$.

(b) $(f,\bar{p})$ and $(\bar{f},\bar{p})$ satisfy CS.

(c) Let $\bar{P}$ be the augmenting path of the iteration. Then

$$R(\bar{p}, \bar{P}) = 0,$$

that is, all arcs of $\bar{P}$ have zero reduced cost with respect to $\bar{p}$.

(d) $\bar{p}_j = p_j$ for all j with $g_j < 0$.

By Prop. 1, if initially f is integer and (f, p) satisfy CS, the same is true after all subsequent iterations. Then at each iteration, the total absolute surplus $\sum_{i \in \mathcal{N}} |g_i|$ will be reduced by the positive integer 2δ, where δ is the augmentation increment given by Eq. (5). Thus only a finite number of reductions of $\sum_{i \in \mathcal{N}} |g_i|$ can occur, so the algorithm must terminate in a finite number of iterations if the problem is feasible.

The following parallel synchronous version of the primal-dual algorithm is a direct generalization of the assignment algorithm of [BMP89]. It starts with a pair (f, p) satisfying CS and generates another pair $(\bar{f}, \bar{p})$ as follows:

Parallel Synchronous Primal-Dual Iteration:

Choose a subset $I = \{i_1, \ldots, i_m\}$ of nodes with positive surplus. (If all nodes have nonpositive surplus, the algorithm terminates.) For each i_n, $n = 1, \ldots, m$, let $\bar{p}(n)$ and $\bar{P}(n)$ be the price vector and augmenting path obtained by executing a primal-dual iteration starting at i_n, and using the pair (f, p). Then generate sequentially the pairs $\big(f(n), p(n)\big)$, $n = 1, \ldots, m$, as follows, starting with $\big(f(0), p(0)\big) = (f, p)$:

For $n = 0, \ldots, m-1$, if $\bar{P}(n+1)$ is an augmenting path with respect to $f(n)$, obtain $f(n+1)$ by augmenting $f(n)$ along $\bar{P}(n+1)$, and set

$$p_j(n+1) = \max\big\{p_j(n), \bar{p}_j(n)\big\}, \qquad \forall\, j \in \mathcal{N}.$$

Otherwise set

$$f(n+1) = f(n), \qquad p(n+1) = p(n).$$

The pair $(\bar{f}, \bar{p})$ generated by the iteration is

$$\bar{f} = f(m), \qquad \bar{p} = p(m).$$

The preceding algorithm can be parallelized by using multiple processors to compute the augmenting paths of an iteration in parallel. On the other hand the algorithm is synchronous in that iterations have clear "boundaries". In particular, all augmenting paths generated in the same iteration are computed on the basis of the same pair (f, p). Thus, it is necessary to synchronize the parallel processors at the beginning of each iteration, with an attendant synchronization penalty.

The parallel asynchronous primal-dual algorithm tries to reduce the synchronization penalty by "blurring" the boundaries between iterations and by allowing processors to compute augmenting paths using pairs (f,p) which are out-of-date.

To describe the parallel asynchronous algorithm, let us denote the flow-price pair at the times

$$k = 1, 2, 3, \ldots$$

by $\big(f(k), p(k)\big)$. (In a practical setting, the times k represent "event times", that is, times at which an attempt is made to modify the pair (f,p) through an iteration.) We require that the initial pair $\big(f(1), p(1)\big)$ satisfies CS. The algorithm terminates when during an iteration, either a feasible flow is obtained or else infeasibility is detected.

k^{th} *Asynchronous Primal-Dual Iteration:*

At time k, a primal-dual iteration is performed on a pair $\big(f(\tau_k), p(\tau_k)\big)$, where τ_k is a positive integer with $\tau_k \leq k$, to produce a pair $\big(\bar{f}(k), \bar{p}(k)\big)$ and an augmenting path $\bar{P}_k$. The iteration (and the path $\bar{P}_k$) is said to be *incompatible* if $\bar{P}_k$ is not an augmenting path with respect to $f(k)$; in this case we discard the results of the iteration, that is, we set

$$f(k+1) = f(k), \qquad p(k+1) = p(k).$$

Otherwise, we say that the iteration (and the path $\bar{P}_k$) is *compatible*, we obtain $f(k+1)$ from $f(k)$ by augmenting $f(k)$ along $\bar{P}_k$, and we set

$$p_j(k+1) = \max\big\{p_j(k), \bar{p}_j(k)\big\}, \qquad \forall\, j \in \mathcal{N}. \tag{14}$$

We note that the definition of the asynchronous algorithm is not yet rigorous, because we have not yet proved that $\big(f(k), p(k)\big)$ satisfies CS at all times prior to termination, so that a primal-dual iteration can be performed. This will be shown in the next section.

The implementation of the asynchronous algorithm in a parallel shared memory machine is quite straightforward. The main idea is to maintain a "master" copy of the current flow-price pair in the shared memory; this is the pair $\big(f(k), p(k)\big)$ in the prededing mathematical description of the algorithm. To execute an iteration, a processor copies from the shared memory the current master flow-price pair; during this copy operation the master pair is locked, so no other processor can modify it. The processor performs a primal-dual iteration using the copy obtained, and then locks the master pair (which may by now differ from the copy obtained earlier). The processor checks if the iteration is compatible, and if so it modifies accordingly the master flow-price pair. The processor then unlocks the master pair, possibly after retaining a copy to use at a subsequent iteration. The times when the master pair is copied and modified by processors correspond to the indexes τ_k and

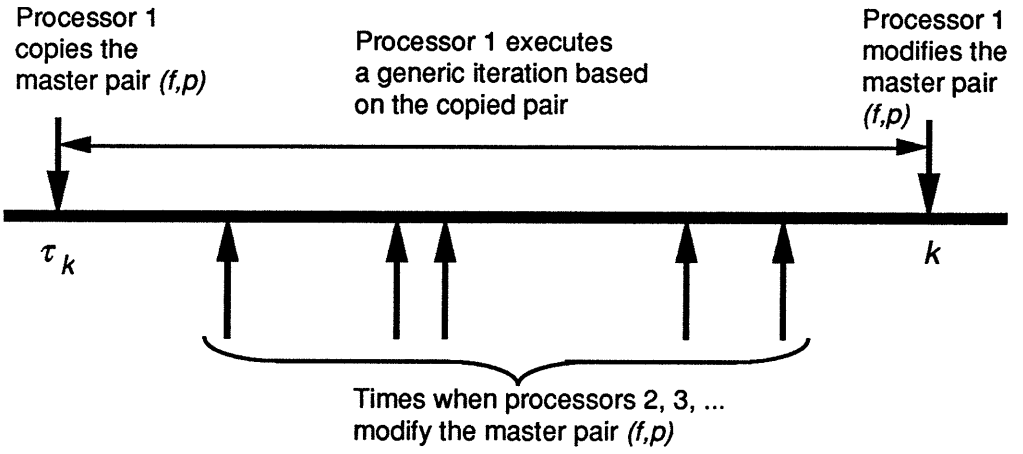

Figure 1: Operation of the asynchronous algorithm in a shared memory machine. A processor copies the master flow-price pair at time τ_k, executes between times τ_k and k a generic iteration using the copy, and modifies accordingly the master flow-price pair at time k. Other processors may have modified unpredictably the master pair between times τ_k and k.

k of the asynchronous algorithm, respectively, as illustrated in Fig. 1. For specific implementation details in the case of an assignment problem, we refer the reader to [BeC90].

We finally note that any sequence of flow-price pairs generated by the synchronous parallel algorithm can also be viewed as a sequence $\big(f(k), p(k)\big)$ generated by the asynchronous version; for this, the time τ_k should correspond to the flow-price pair available at the start of the synchronous algorithm iteration during which $\big(f(k), p(k)\big)$ was generated. Thus, our subsequent proof of validity of the asynchronous algorithm applies also to the synchronous version.

3. VALIDITY OF THE ASYNCHRONOUS ALGORITHM

We want to show that the asynchronous algorithm maintains CS throughout its course. We first introduce some definitions and then we break down the main argument of the proof in a few lemmas.

Lemma 1: Assume that (f,p) satisfies CS. Let $P = (i_1, i_2, \ldots, i_k)$ be an unblocked path with respect to f. Then

$$p_{i_k} = p_{i_1} + R(p,P) - C(P).$$

Proof: Using Eqs. (7) and (8), we have

$$\begin{aligned} R(p,P) &= \sum_{(i_m,i_{m+1})\in P^+} (a_{i_m i_{m+1}} + p_{i_{m+1}} - p_{i_m}) - \sum_{(i_{m+1},i_m)\in P^-} (a_{i_{m+1} i_m} + p_{i_m} - p_{i_{m+1}}) \\ &= C(P) + \sum_{m=1}^{k-1} (p_{i_{m+1}} - p_{i_m}) \\ &= C(P) + p_{i_k} - p_{i_1}, \end{aligned}$$

which yields the desired result. **Q.E.D.**

Lemma 2: Let $g_j(k)$ denote the surplus of node j corresponding to $f(k)$. For all nodes j such that $g_j(k) < 0$, we have $p_j(k+1) = p_j(t)$ for all $t \leq k$.

Proof: By the nature of augmentations, we have $g_j(t) \leq g_j(t+1) \leq 0$ if $g_j(t) < 0$. Therefore, the hypothesis implies that $g_j(t) < 0$ for all $t \leq k$ and the result follows from Prop. 1(d). **Q.E.D.**

Lemma 3: Let $k \geq 1$ be given and assume that $\big(f(t), p(t)\big)$ satisfies CS for all $t \leq k$. Then:

(a) For all nodes j and all $t \leq k$, there holds

$$\bar{p}_j(t) \leq p_j(\tau_t) + d_j(\tau_t). \tag{15}$$

(b) For $t \leq k$, if $f(t+1) \neq f(t)$ (i.e., iteration t is compatible), and j is a node which belongs to the corresponding augmenting path, then we have

$$p_j(t) + d_j(t) = \bar{p}_j(t) = p_j(t+1). \tag{16}$$

(c) For all nodes j and all $t \leq k-1$, there holds

$$p_j(t) + d_j(t) \leq p_j(t+1) + d_j(t+1). \tag{17}$$

Proof: (a) If j is such that $g_j(\tau_t) < 0$, by Prop. 1(d), we have $\bar{p}_j(t) = p_j(\tau_t)$ and $d_j(t) = 0$, so the result holds. Thus, assume that $g_j(\tau_t) \geq 0$. Consider any unblocked path P from j to a node $\overline{j}$ with $g_{\overline{j}}(\tau_t) < 0$. By Lemma 1, we have

$$p_{\overline{j}}(\tau_t) = p_j(\tau_t) + R\big(p(\tau_t), P\big) - C(P),$$

$$\bar{p}_{\overline{j}}(t) = \bar{p}_j(t) + R\big(\bar{p}(t), P\big) - C(P).$$

Since $g_{\overline{j}}(\tau_t) < 0$, we have $p_{\overline{j}}(\tau_t) = \bar{p}_{\overline{j}}(t)$ and it follows that

$$\bar{p}_j(t) = p_j(\tau_t) + R\big(p(\tau_t), P\big) - R\big(\bar{p}(t), P\big) \leq p_j(\tau_t) + R\big(p(\tau_t), P\big).$$

Taking the minimum of $R\big(p(\tau_t), P\big)$ over all unblocked paths P, starting at j and ending at nodes $\overline{j}$ with $g_{\overline{j}}(\tau_t) < 0$, the result follows.

(b), (c) We prove parts (b) and (c) simultaneously, by first proving a weaker version of part (b) (see Eq. (18) below), then proving part (c), and then completing the proof of part (b). Specifically, we will first show that for $t \leq k$, if $f(t+1) \neq f(t)$ and j is a node which belongs to the corresponding augmenting path, then we have

$$p_j(t) + d_j(t) \leq \bar{p}_j(t) = p_j(t+1). \tag{18}$$

Indeed, if $g_j(t) < 0$, Eq. (18) holds since, by Lemma 2, we have $p_j(t) = \bar{p}_j(t)$ and $d_j(t) = 0$. Assume that $g_j(t) \geq 0$. Let the augmenting path of iteration t end at node $\overline{j}$, and let P be the portion of this path that starts at j and ends at $\overline{j}$. We have, using Lemma 1 and Prop. 1(c),

$$\bar{p}_{\overline{j}}(t) = \bar{p}_j(t) + C(P),$$

$$p_{\overline{j}}(t) = p_j(t) + C(P) + R\big(p(t), P\big).$$

Since $g_{\overline{j}}(t) < 0$, by Lemma 2, we have $\bar{p}_{\overline{j}}(t) = p_{\overline{j}}(t)$, and we obtain

$$\bar{p}_j(t) = p_j(t) + R\big(p(t), P\big) \geq p_j(t) + d_j(t),$$

showing the left hand side of Eq. (18). Since $d_j(t) \geq 0$, this yields $p_j(t) \leq \bar{p}_j(t)$, so $\bar{p}_j(t) = \max\{p_j(t), \bar{p}_j(t)\} = p_j(t+1)$, completing the proof of Eq. (18).

We now prove part (c), making use of Eq. (18). Let us fix node j and assume without loss of generality that iteration t is compatible (otherwise Eqs. (16) and (17) hold trivially). If $g_j(t+1) < 0$, we have $p_j(t) = p_j(t+1)$ and $d_j(t) = d_j(t+1) = 0$, so the desired relation (17) holds. Thus, assume that $g_j(t+1) \geq 0$, and let $P = (j, j_1, \ldots, j_k, \overline{j})$ be an unblocked path with respect to $f(t+1)$, which is such that $g_{\overline{j}}(t+1) < 0$ and

$$R\big(p(t+1), P\big) = d_j(t+1).$$

There are three possibilities:

(1) The nodes $j, j_1, \ldots, j_k$ do not belong to the augmenting path of iteration t. In this case, the path P is also unblocked with respect to $f(t)$. By using Lemma 1, it follows that

$$p_{\overline{j}}(t+1) = p_j(t+1) + C(P) + R\big(p(t+1), P\big),$$

and

$$p_{\overline{j}}(t) = p_j(t) + C(P) + R\big(p(t), P\big).$$

Since $g_{\bar{j}}(t+1) < 0$, we have $p_{\bar{j}}(t+1) = p_{\bar{j}}(t)$, so the preceding equations yield

$$p_j(t+1) + R\big(p(t+1), P\big) = p_j(t) + R\big(p(t), P\big).$$

Since $R\big(p(t+1), P\big) = d_j(t+1)$ and $R\big(p(t), P\big) \geq d_j(t)$, we obtain

$$p_j(t) + d_j(t) \leq p_j(t+1) + d_j(t+1),$$

and the desired relation (17) is proved in this case.

(2) Node j belongs to the augmenting path of iteration t, in which case, by Eq. (18), we have

$$p_j(t) + d_j(t) \leq p_j(t+1) \leq p_j(t+1) + d_j(t+1),$$

and the desired relation (17) is proved in this case as well.

(3) There is a node j_m, $m \in \{1, \ldots, k\}$, which belongs to the augmenting path of iteration t, and is such that j and $j_1, \ldots, j_{m-1}$ do not belong to the augmenting path of iteration t. Consider the following unblocked paths with respect to $f(t+1)$

$$P' = (j, j_1, \ldots, j_{m-1}, j_m),$$
$$P'' = (j_m, j_{m+1}, \ldots, j_k, \bar{j}).$$

By using Lemma 1, we have

$$R\big(p(t+1), P'\big) + p_j(t+1) = R\big(p(t), P'\big) + p_j(t) + p_{j_m}(t+1) - p_{j_m}(t),$$

and since by Eq. (18), $p_{j_m}(t+1) - p_{j_m}(t) \geq d_{j_m}(t)$, we obtain

$$R\big(p(t+1), P'\big) + p_j(t+1) \geq R\big(p(t), P'\big) + p_j(t) + d_{j_m}(t). \tag{19}$$

On the other hand, we have

$$R\big(p(t+1), P\big) = R\big(p(t+1), P'\big) + R\big(p(t+1), P''\big)$$

and since $R\big(p(t+1), P''\big) \geq 0$, we obtain

$$R\big(p(t+1), P\big) \geq R\big(p(t+1), P'\big). \tag{20}$$

Combining Eqs. (19) and (20), we see that

$$R\big(p(t+1), P\big) + p_j(t+1) \geq R\big(p(t), P'\big) + p_j(t) + d_{j_m}(t).$$

We have $R\big(p(t), P'\big) + d_{j_m}(t) \geq d_j(t)$, and $R\big(p(t+1), P\big) = d_j(t+1)$, so it follows that

$$p_j(t+1) + d_j(t+1) \geq p_j(t) + d_j(t),$$

and the proof of part (c) is complete.

To complete the proof of part (b), we note that by using Eqs. (15) and (17), we obtain

$$\bar{p}_j(t) \leq p_j(\tau_t) + d_j(\tau_t) \leq p_j(t) + d_j(t),$$

which combined with Eq. (18) yields the desired Eq. (17). **Q.E.D.**

We can now prove that the asynchronous algorithm preserves CS.

Proposition 2: All pairs $(f(k), p(k))$ generated by the asynchronous algorithm satisfy CS.

Proof: By induction. Suppose all iterations up to the k^{th} maintain CS, let the k^{th} iteration be compatible, and let $\bar{P}_k$ be the corresponding augmenting path. We will show that the pair $(f(k+1), p(k+1))$ satisfies CS. For any arc (i,j) there are three possibilities:

(1) $f_{ij}(k+1) \neq f_{ij}(k)$. In this case by Prop. 1(c), we have $\bar{p}_i(k) = a_{ij} + \bar{p}_j(k)$. Since i and j belong to $\bar{P}_k$, by Lemma 3(b), we have $p_i(k+1) = \bar{p}_i(k)$ and $p_j(k+1) = \bar{p}_j(k)$, implying that $p_i(k+1) = a_{ij} + p_j(k)$, so the CS condition is satisfied for arc (i,j).

(2) $f_{ij}(k+1) = f_{ij}(k) < c_{ij}(k)$. In this case, by the CS property (cf. the induction hypothesis), we have $p_i(k) \leq a_{ij} + p_j(k)$. If $p_i(k) \geq \bar{p}_i(k)$, it follows that

$$p_i(k+1) = p_i(k) \leq a_{ij} + p_j(k) \leq a_{ij} + p_j(k+1),$$

so the CS condition is satisfied for arc (i,j). Assume therefore that $p_i(k) < \bar{p}_i(k)$. If $f_{ij}(\tau_k) < c_{ij}(\tau_k)$, then since by Prop. 1(b), $(f, \bar{p})$ satisfies CS, we have $\bar{p}_i(k) \leq a_{ij} + \bar{p}_j(k)$, from which $p_i(k) \leq a_{ij} + \bar{p}_j(k) \leq a_{ij} + p_j(k+1)$, and again the CS condition is satisfied for arc (i,j). The last remaining possibility (under the assumption $f_{ij}(k+1) = f_{ij}(k) < c_{ij}(k)$) is that $f_{ij}(\tau_k) = c_{ij}(\tau_k)$ and $p_i(k) < \bar{p}_i(k)$. We will show that this can't happen by assuming that it does and then arriving at a contradiction. Let t_1 be the first time index such that $\tau_k < t_1 \leq k$ and $f_{ij}(t_1) < c_{ij}(t_1)$. Then by Lemmas 3(a) and 3(c), we have

$$\bar{p}_i(k) \leq p_i(\tau_k) + d_i(\tau_k) \leq p_i(t_1 - 1) + d_i(t_1 - 1),$$

while from Lemma 3(b),

$$p_i(t_1 - 1) + d_i(t_1 - 1) = p_i(t_1) \leq p_i(k),$$

(since $f_{ij}(t_1) \neq f_{ij}(t_1 - 1)$ and node i belongs to the augmenting path of iteration $t_1 - 1$). It follows that $\bar{p}_i(k) \leq p_i(k)$, which contradicts the assumption $p_i(k) < \bar{p}_i(k)$, as desired. We have thus shown that the CS condition holds for arc (i,j) in case (2).

(3) $f_{ij}(k+1) = f_{ij}(k) > b_{ij}(k)$. The proof that the CS condition is satisfied for arc (i,j) is similar as for the preceding case (2).

Q.E.D.

Proposition 2 shows that if the asynchronous algorithm terminates, the assignment-price pair obtained satisfies CS. Since the assignment obtained at termination is complete, it must be optimal. To guarantee that the algorithm terminates, we impose the condition

$$\lim_{k\to\infty} \tau_k = \infty.$$

This is a natural and essential condition, stating that the algorithm iterates with increasingly more recent information.

Proposition 3: If $\lim_{k\to\infty} \tau_k = \infty$, the asynchronous algorithm terminates. If the problem is feasible, the flow obtained at termination is optimal.

Proof: There can be at most a finite number of compatible iterations, so if the algorithm does not terminate, all iterations after some index $\bar{k}$ are incompatible, and $f(k) = f(\bar{k})$ for all $k \geq \bar{k}$. On the other hand, since $\lim_{k\to\infty} \tau_k = \infty$, we have that $\tau_k \geq \bar{k}$ for all k sufficiently large, so that $f(\tau_k) = f(k)$ for all $k \geq \bar{k}$. This contradicts the incompatibility of the k^{th} iteration. **Q.E.D.**

4. COMBINATION WITH SINGLE NODE RELAXATION ITERATIONS

Computational experiments show that in a serial setting, primal-dual methods are greatly speeded up by mixing shortest path augnmentations with single node relaxation (or coordinate ascent) iterations of the type introduced in [Ber82]. The typical single node iteration starts with a pair (f,p) satisfying CS and produces another pair $(\bar{f},\bar{p})$ satisfying CS. It has the following form.

Single Node Relaxation Iteration:

Choose a node i with $g_i > 0$. Let

$$B_i^+ = \{j \mid (i,j) \in \mathcal{A},\ r_{ij} = 0,\ f_{ij} < c_{ij}\},$$

$$B_i^- = \{j \mid (j,i) \in \mathcal{A},\ r_{ji} = 0,\ f_{ji} > b_{ji}\}.$$

Step 1: If

$$g_i \geq \sum_{j\in B_i^+} (c_{ij} - f_{ij}) + \sum_{j\in B_i^-} (f_{ji} - b_{ji}),$$

go to Step 4. Otherwise, choose a node $j \in B_i^+$ with $g_j < 0$ and go to Step 2, or choose a node $j \in B_i^-$ with $g_j < 0$ and go to Step 3; if no such node can be found, set $\bar{f} = f$ and $\bar{p} = p$, and terminate the iteration.

Step 2: (Flow Adjustment on Outgoing Arc) Let

$$\delta = \min\{g_i, -g_j, c_{ij} - f_{ij}\}.$$

Set

$$f_{ij} := f_{ij} + \delta, \qquad g_i := g_i - \delta, \qquad g_j := g_j + \delta$$

and go to Step 1.

Step 3: (Flow Adjustment on Incoming Arc) Let

$$\delta = \min\{g_i, -g_j, f_{ji} - b_{ji}\}.$$

Set

$$f_{ji} := f_{ji} - \delta, \qquad g_i := g_i - \delta, \qquad g_j := g_j + \delta$$

and go to Step 1.

Step 4: (Increase Price of i) Set

$$g_i := g_i - \sum_{j \in B_i^+} (c_{ij} - f_{ij}) - \sum_{j \in B_i^-} (f_{ji} - b_{ji}),$$

$$f_{ij} = c_{ij}, \qquad \forall\, j \in B_i^+,$$

$$f_{ji} = b_{ji}, \qquad \forall\, j \in B_i^-,$$

$$p_i := \min\Big\{\min\{p_j + a_{ij} \mid (i,j) \in \mathcal{A},\ p_i < p_j + a_{ij}\}, \min\{p_j - a_{ji} \mid (j,i) \in \mathcal{A},\ p_i < p_j - a_{ji}\}\Big\}.$$

If following these changes $g_i > 0$, recalculate the sets B_i^+ and B_i^+, and go to Step 1; else, set $\bar{f} = f$ and $\bar{p} = p$, and terminate the iteration.

It can be sheen that the flow changes of the above iteration are such that the condition $g_i \geq 0$ is maintained. Furthermore, it can be shown that the pair $(\bar{f}, \bar{p})$ generated by the iteration satisfies CS. To see this, first note that Steps 2 and 3 can only change flows of arcs with zero reduced cost; then observe that the flow changes in Step 4 are designed to maintain CS of the arcs whose reduced cost changes from zero to nonzero, and the price change is such that the sign of the reduced costs of all other arcs does not change from positive to negative or reversely.

A combined primal-dual/single node relaxation iteration can now be constructed. It starts with a pair (f, p) satisfying CS and produces another pair $(\bar{f}, \bar{p})$ as follows:

Combined Primal-Dual/Relaxation Iteration:

Choose a node i with $g_i > 0$ (if no such node can be found, stop the algorithm). Perform a single node relaxation iteration. If as a result (f, p) is changed, terminate the iteration; otherwise, perform a primal-dual iteration starting from (f, p).

A synchronous parallel combined method can be constructed based on the above iteration. To this end, we must modify the definition of compatibility for the case where the pair $(\bar{f}(n), \bar{p}(n))$ (refer to the description of the synchronous parallel iteration in Section 2) is produced by the single node relaxation iteration. In this case, we discard the results of the iteration if

$$\bar{p}_{i_n}(n) < p_{i_n}(n),$$

where i_n is the node i used in the single node iteration. Otherwise, we say that the iteration is *compatible*, we set

$$p_i(n+1) = \begin{cases} \bar{p}_{i_n} & \text{if } i = i_n, \\ p_i(n) & \text{otherwise,} \end{cases}$$

and for all arcs (i, j), we set

$$f_{ij}(n+1) = \begin{cases} f_{ij}(n) & \text{if } i \neq i_n \text{ and } j \neq i_n, \\ \bar{f}_{ij}(n) & \text{if } i = i_n \text{ or } j \neq i_n, \text{ and } r_{ij}(n+1) = 0, \\ b_{ij} & \text{if } i = i_n \text{ or } j \neq i_n, \text{ and } r_{ij}(n+1) > 0, \\ c_{ij} & \text{if } i = i_n \text{ or } j \neq i_n, \text{ and } r_{ij}(n+1) < 0, \end{cases}$$

where $r_{ij}(n+1)$ is the reduced cost of arc (i, j) with respect to the price vector $p(n+1)$.

The definition of compatibility is such that the above synchronous parallel iteration preserves CS. Using this property and the monotonic increase of the node prices, it can be seen that the associated algorithm terminates finitely, assuming the problem is feasible. A similar result can be shown for the corresponding asynchronous version of the parallel iteration.

5. COMPUTATIONAL RESULTS

In order to illustrate the expected performance of the above parallel primal dual minimum cost network flow algorithms, we designed a synchronous parallel version of one of the primal dual codes developed by Bertsekas and Tseng for comparison with the RELAX code (see [BeT85] for a description). We implemented this synchronous parallel primal dual on a shared-memory Encore Multimax and evaluated the parallel computation time for two minimum cost transshipment problems as a

function of the number of processors used. In this section, we briefly overview this parallel implementation and discuss the numerical results obtained.

The algorithm operates as follows: Each iteration starts synchronously with each processor copying the current set of node prices and arc flows (f,p) satisfying CS. Each processor $n = 1,\ldots,m$ selects a different node i_n with positive surplus, and performs a primal dual iteration to compute a shortest augmenting path (in terms of the reduced cost lengths) from node i_n to the set of nodes with negative surplus. Let $\bar{p}(n)$ and $\bar{P}(n)$ be the price vector and augmenting path obtained by processor n.

Assume without loss of generality that the m processors find their shortest augmenting paths in the order $n = 1,\ldots,m$, and let $\big(f(n),p(n)\big)$ denote the flow-price vector pair resulting from incorporation of the results of the processor n (note that $\big(f(0),p(0)\big) = (f,p)$). As described in Section 2, once a processor computes $\bar{p}(n)$ and $\bar{P}(n)$, it checks to see whether $\bar{P}(n)$ is a compatible augmentation based on the most recent network prices and flows $\big(f(n-1),p(n-1)\big)$. During this operation, the network is locked so that only one processor (at a time) can verify the compatibility of an augmentation or modify the flow-price vector pair. If the augmentation is compatible, the arc flows are modified accordingly and the node prices are adjusted as described in Section 2. The processor then waits for all other processors to complete their computations before starting the next cycle of augmentations.

In our implementation on the Encore Multimax, the most recent flow-price vector pair $\big(f(n),p(n)\big)$ is kept in shared memory; in addition, each processor copies to its local memory the pair $(f,p) = \big(f(0),p(0)\big)$ stored in shared memory at the beginning of the iteration. The set of nodes with positive surplus is maintained in a queue; a lock on this queue is used in order to guarantee that a given node can be selected by only one processor. A synchronization lock on the flow-price vector pair $\big(f(n),p(n)\big)$ is used to restrict modifications of flows of prices by more than one processor simultaneously, and a synchronization barrier is used at the end of each iteration to synchronize the next iteration.

The principal drawback of our implementation of the synchronous algorithm is the idle time spent by the processors waiting while other processors are still computing augmenting paths or modifying the pair $\big(f(n),p(n)\big)$ that is kept in shared memory. Figure 2 illustrates the processor idle times in a typical iteration.

Table 1 illustrates the performance of the algorithm on the Encore Multimax for two uncapacitated transshipment problems generated using the widely used NETGEN program of [KNS74]; these problems correspond to problems NG31 and NG35 in [KNS74]. Problem NG31 has 1000 nodes and 4800 arcs, with 50 sources and 50 sinks, while problem NG35 has 1500 nodes and 5730 arcs, with 75 sources and 75 sinks. The table contains the time required for the algorithm in three different runs,

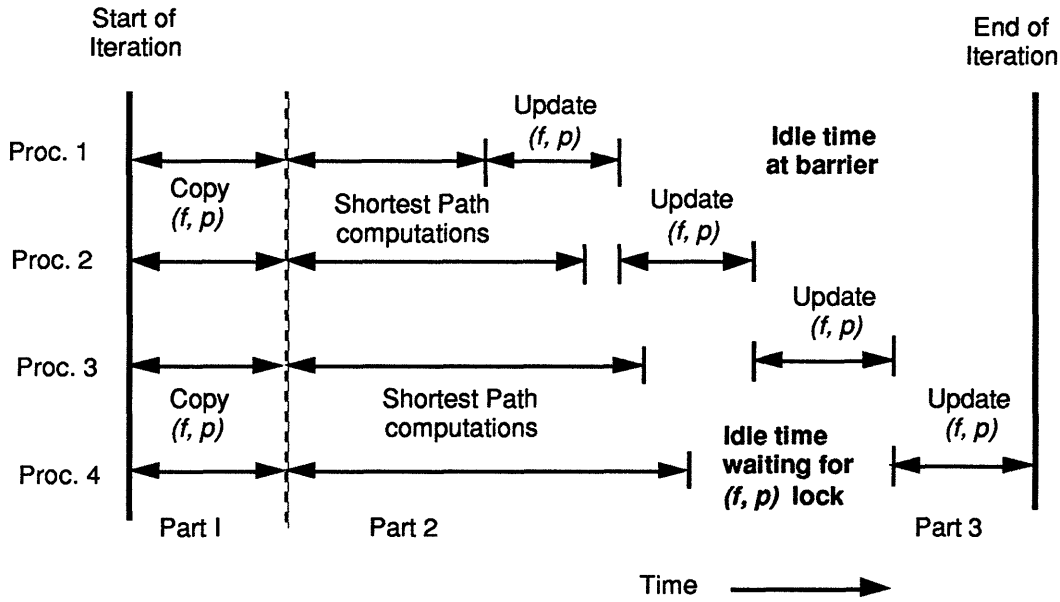

Figure 2: Timing diagram of an iteration. The computation of each processor consists of three parts, possibly separated by idle time. In the first part, all processors copy (in parallel) the master pair (f, p). In the second part, the processors calculate (in parallel) their shortest augmenting paths. In the third part, the processors update (one-at-a-time) the master pair (f, p). The next iteration does not begin until all processors have finished all three parts.

as a function of the number of processors used. Note the variability of the run times for different runs; this is due to randomness in the order of completion of the computations of the individual processors, which can lead to differences as to which augmentations are found compatible.

Problem	# of Processors	Run # 1	Run # 2	Run # 3
NG31	1	22.85	22.89	22.83
	2	16.26	15.65	16.03
	3	13.60	13.18	13.07
	4	14.00	13.98	14.02
NG35	1	54.33	53.98	54.10
	2	37.43	39.14	37.67
	3	33.58	31.17	29.72
	4	31.10	28.85	29.39

Table 1: Run times in secs on the Encore Multimax for problems NG31 and NG35 of [KNS74].

Table 1 suggests that the algorithm can achieve a rather limited speedup. There are two primary reasons for this: a) the synchronization overhead, that is, the processor idle time per iteration illustrated in Fig. 2, and b) the nearly sequential part of the computation near convergence when there are very few nodes with positive surplus. Similar limitations were observed in [BeC90] in the context of parallel Hungarian algorithms for assignment problems. The performance of the algorithm observed in Table 1 is actually better than the results reported for the comparable synchronous parallel algorithm reported in [BeC90]. For a more detailed discussion of these limiting factors, the reader is referred to [BeC90], where extensive numerical experiments were reported which measured both the synchronization overhead and the sequential part of the computation.

Alternative parallel algorithms which significantly reduce the synchronization overhead can be designed using the theory described in Sections 2 and 3. One approach is to have each processor search for multiple augmenting paths (from different starting nodes with positive surplus) during each iteration. In this manner, the number of iterations is considerably reduced, thereby reducing the overall synchronization overhead. To make this approach efficient, the assignment of positive surplus nodes to each processor should be adaptive, depending on the time required to find the previous augmentations. Such an algorithm was implemented and evaluated in [BeC90] in the context of assignment problem, yielding significant reductions in synchronization overhead. A second approach to reducing the synchronization overhead is to implement an asynchronous parallel primal-dual algorithm, based on the algorithm described in Section 2. Such an algorithm would allow processors to start new shortest path computations without waiting for other processors to complete their shortest path computations. Again, experiments reported in [BeC90] in the context of assignment problems indicate that significant reductions in synchronization overhead can be achieved through this approach.

REFERENCES

[AMO89] Ahuja, R. K., Magnanti, T. L., and Orlin, J. B., "Network Flows", Sloan W. P. No. 2059-88, M.I.T., Cambridge, MA, March 1989 (also in Encyclopedia of Operations Research).

[BMP89] Balas, E., Miller, D., Pekny, J., and Toth, P., " A Parallel Shortest Path Algorithm for the Assignment Problem", Management Science Report MSRR 552, Carnegie Mellon Univ., Pittsburgh, PA, April 1989.

[BeC89] Bertsekas, D. P., and Castañon, D. A., "Parallel Synchronous and Asynchronous Implementations of the Auction Algorithm", Alphatech Report, Nov. 1989, submitted for publication.

[BeC70] Bertsekas, D. P., and Castañon, D. A., "Parallel Asynchronous Hungarian Methods for the Assignment Problem", Alphatech Report, Feb. 1990, submitted for publication.

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

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

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

[BuG61] Busaker, R. G., and Gowen, P. J., "A Procedure for Determining a Family od Minimal-Cost Network Flow Patterns", O.R.O. Technical Report No. 15, Operational Research Office, John Hopkins University, Baltimore, MD, 1961.

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

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

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

[Law76] Lawler, E., "Combinatorial Optimization: Networks and Matroids", Holt, Rinehart and Winston, New York, 1976.

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

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