On the Massively Parallel Solution of Linear Network Flow Problems

SOREN S. NIELSEN, STAVROS A. ZENIOS

February 1992
Revised August 1992

Abstract

We report on a comparative computational study of two algorithms for the massively parallel solution of linear network problems: the ϵ-relaxation algorithm, and the Proximal Minimization algorithm with D-functions. Problems with up to 16 million variables are solved in about 1.5 hours. The massively parallel algorithms are compared with simplex and relaxation codes on vector supercomputers as well as small-scale parallel machines.

1. Introduction

In this paper, we report on a comparative computational study of two algorithms for the massively parallel solution of *linear* network programs: the ϵ-relaxation algorithm of Bertsekas [1986a, 1986b] and the Proximal Minimization algorithm with D-functions (PMD) of Censor and Zenios [1992]. We also compare the massively parallel algorithms with solution techniques on vector supercomputers as well as small-scale parallel machines. Network flow problems with *strictly convex*, separable objective functions have been solved efficiently on massively parallel computers by solving the dual problem. For the strictly convex problem, the dual functional is differentiable, and the dual problem can be solved using row-action methods or simple dual ascent methods. The success of the massively

1991 Mathematics Subject Classification. Primary 90B10, 65Y05.
Partially supported by NSF grant CCR–9104042 and AFOSR grant 91–0168.
Final version. No other version will be submitted for publication elsewhere.

parallel implementations is due to the fact that dual ascent steps can be taken concurrently along all dual coordinate directions.

Differentiability of the dual functional does not, however, hold in the case of the linear network problem. Here, situations arise where dual ascent cannot be achieved by steps along any single dual coordinate direction. Steps must be taken along multiple coordinates, and this destroys the parallel decomposition nature of dual ascent algorithms.

Two algorithms to overcome this difficulty have been suggested. The ϵ-relaxation algorithm deals with the non-differentiability of the dual functional by maintaining only approximate complementary slackness. Steps are taken along a dual coordinate direction even if they result in a small deterioration in the dual objective value. The algorithm is, for integral problem data, finitely convergent. The PMD algorithm solves the linear program by introducing a non-linear objective perturbation, namely a certain directed distance measure to a "proximal point". This results in having to solve a sequence of strictly convex subproblems, which can be done using the row-action algorithm of Censor and Lent [1981]. By using a quadratic and entropic distance measure, respectively, we get as special cases of the PMD algorithm the quadratic proximal point algorithm (QPP) of Rockafellar [1976a, 1976b], and the entropic proximal point (EPP) algorithm of Eriksson [1985].

Common for these algorithms is that they decompose in such a way that dual steps, corresponding to updates of the dual prices associated with each network node, can be performed in a massively parallel fashion. By operating on all network nodes concurrently, highly parallelizable Jacobi-type algorithms are obtained (Jacobi-type algorithms, in contrast to Gauss-Seidel algorithms, may operate at each iteration on data which are several steps old. They have poorer convergence properties, but in a parallel implementation they typically have less communication and synchronization overheads).

The dual prices are, in the case of dual relaxation algorithms, updated using a linesearch procedure, Nielsen and Zenios [1992b]. In the case of the row-action algorithm, a simultaneous primal-dual update is performed using a *Bregman projection*, i.e., a primal projection, using a certain distance measure, onto a constraint hyperplane and a corresponding dual update (for details of the row-action algorithms, see, e.g., Nielsen and Zenios [1992b], Linesearch 1).

The objective of our study is to establish the suitability of massively parallel computers for the solution of huge, linear network problems. The algorithms studied in this paper are implemented on a Connection Machine CM–2 with up to 32K processing elements and problems with up to 16 million variables are solved. To further establish the performance of these algorithms on the CM–2, we compare them to a state-of the art network simplex code executing on a supercomputer, the Cray Y–MP, and to a sequential implementation of the Relax code of Bertsekas [1991]. In order to estimate the efficiency of the parallel implementations, we also compare the PMD algorithms on the CM–2 to a small-scale, parallel implementation of the Gauss-Seidel variant of the same algorithm, and in addition show how the massively parallel algorithm scales with different numbers of CM–2 processors.

This paper was written as part of the DIMACS challenge on the practical implementation of algorithms for network flow problems. However, the computational results have been compiled over the period 1989–1992, as part of ongoing projects on parallel network optimization, and the test problems are not those of DIMACS. Hence, our results are not directly comparable with the results of other studies in the DIMACS challenge, but they are still based on publicly available data.

The paper is organized as follows. In Sections 2.1 and 2.2 we describe the ϵ-relaxation algorithm and the PMD algorithms, respectively. Section 3 is a computational study of the algorithms. We test the performance of the ϵ-relaxation and the PMD algorithms, respectively, in Sections 3.2 and 3.3. In Sections 3.4 and 3.5 we apply the PMD algorithms for the solution of problems with up to 16 million variables, and then investigate the question of scalability of the algorithms on the CM–2. In Section 3.6 we compare the CM–2 implementation of the QPP algorithm with a small-scale, parallel implementation. In Section 4 we compare the ϵ-relaxation and PMD algorithms to each other, and to a state-of-the-art network simplex solver running on a Cray Y–MP. Section 5 summarizes the conclusions of our study.

2. The Linear Network Optimization Algorithms

We consider the directed graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$, where $\mathcal{N}$ is the set of nodes with cardinality m and $\mathcal{E}$ is the set of arcs with cardinality n. Finding a feasible flow at minimum cost on this graph can be formulated as the following linear program:

$$[\textbf{LNFP}] \quad \underset{x \in \Re^n}{\text{Minimize}} \quad \sum_{(i,j) \in \mathcal{E}} c_{ij} x_{ij} \tag{1}$$

$$\text{Subject to} \quad \sum_{\{j \,|\, (i,j) \in \mathcal{E}\}} x_{ij} - \sum_{\{k \,|\, (k,i) \in \mathcal{E}\}} x_{ki} = b_i, \quad i \in \mathcal{N} \tag{2}$$

$$l_{ij} \le x_{ij} \le u_{ij} \qquad (i,j) \in \mathcal{E} \tag{3}$$

where $c = (c_{ij}) \in \Re^n$ is the vector of cost coefficients, $b = (b_i) \in \Re^m$ is the vector of exogenous supplies or demands, and $l = (u_{ij}) \in \Re^n$ and $u = (u_{ij}) \in \Re^n$ are the vectors of lower and upper bounds on the flows. We associate with the equality constraints (flow conservation constraints) the dual prices $\pi = (\pi_i) \in \Re^m$.

A primal/dual pair (x, π) is optimal if x satisfies the flow conservation constraints (2) and (x, π) satisfy the following complementary slackness (CS) conditions:

If $c_{ij} - \pi_i + \pi_j > 0$ then $x_{ij} = l_{ij}$ and arc (i,j) is termed *inactive*.
If $c_{ij} - \pi_i + \pi_j = 0$ then $l_{ij} \le x_{ij} \le u_{ij}$ and arc (i,j) is termed *balanced*.
If $c_{ij} - \pi_i + \pi_j < 0$ then $x_{ij} = u_{ij}$ and arc (i,j) is termed *active*.
We describe next two algorithms for solving this problem.

2.1 The ϵ-Relaxation Algorithm. The ϵ-relaxation algorithm iteratively adjusts the dual prices and computes flows based on small violations (ϵ) of the

CS conditions. The price adjustment for a particular node is chosen such that the computed flows satisfy conservation of flow for that particular node. Since the algorithm allows small violations (ϵ) of the CS conditions, each step is not guaranteed to improve the dual cost. However, it improves the dual cost of a perturbed problem where some of the arc coefficients are modified by a small constant ϵ. It can be shown, see Bertsekas [1979], Bertsekas and Tsitsiklis [1989, Proposition 3.1, p. 357], that if all problem data are integral then a solution that violates CS by $\epsilon < \frac{1}{m}$ is optimal. Furthermore, the ϵ-relaxation algorithm will obtain an exact solution in a finite number of steps. It has been shown – see Goldberg [1986], Goldberg and Tarjan [1987] – that a polynomial variant of the algorithm is possible when ϵ is successively scaled from a large initial value to less than $\frac{1}{m}$.

In order to formally state the algorithm we introduce first the ϵ-CS conditions. For any price π and a scalar constant $\epsilon > 0$ we characterize an arc (i, j) as:

$$\begin{array}{ll}
\epsilon - \text{inactive}, & \text{if } c_{ij} - \pi_i + \pi_j > \epsilon \\
\epsilon^- - \text{balanced}, & \text{if } c_{ij} - p_\pi + \pi_j = \epsilon \\
\epsilon - \text{balanced}, & \text{if } -\epsilon \leq c_{ij} - \pi_i + \pi_j \leq \epsilon \\
\epsilon^+ - \text{balanced}, & \text{if } c_{ij} - \pi_i + \pi_j = -\epsilon \\
\epsilon - \text{active}, & \text{if } c_{ij} - \pi_i + \pi_j < -\epsilon.
\end{array}$$

The ϵ-CS conditions (ϵ-CS) for a primal/dual pair (x, π) are defined as:

$$\begin{array}{ll}
x_{ij} = l_{ij} & \text{if arc } (i,j) \text{ is } \epsilon - \text{inactive} \\
l_{ij} \leq x_{ij} \leq u_{ij} & \text{if arc } (i,j) \text{ is } \epsilon - \text{balanced} \\
x_{ij} = u_{ij} & \text{if arc } (i,j) \text{ is } \epsilon - \text{active}.
\end{array}$$

The ϵ-relaxation algorithm iteratively chooses nodes that violate conservation of flow and adjust the incident flows to maintain ϵ-CS while reducing the violation. If such a flow adjustment is not possible then a price adjustment takes place. Violations of the flow conservation constraints are measured as the surplus of a node defined by

$$g_i = b_i - \sum_{\{j \mid (i,j) \in \mathcal{E}\}} x_{ij} + \sum_{\{k \mid (k,i) \in \mathcal{E}\}} x_{ki}. \tag{4}$$

Flow adjustments on an arc are possible only if one of the following conditions are satisfied:

1. The arc is ϵ^+-balanced and $x_{ij} < u_{ij}$. In this case a non-zero increase of flow is possible, without violating ϵ-CS. This arc is ϵ-unblocked.

2. The arc is ϵ^--balanced and $x_{ij} > l_{ij}$. In this case a non-zero decrease of flow is possible, without violating ϵ-CS. This arc is ϵ^--unblocked.

We have now all the components needed to define the basic iterative step of ϵ-relaxation for nodes with positive surplus.

Positive Surplus Up Iteration:

Step 0 Let (x, π) satisfy ϵ-CS, and let i be a node with $g_i > 0$.

Step 1 (Scan arcs incident to node i).
Select a node j such that (i,j) is ϵ^+-unblocked and go to Step 2, or select a node j such that (j,i) is ϵ^--unblocked and go to Step 3. If no unblocked arc can be found go to Step 4.

Step 2 (Decrease surplus of node i by increasing x_{ij} on outgoing arcs).
Let $\delta = \min\{g_i, u_{ij} - x_{ij}\}$, update x_{ij}, g_i and g_j as follows:

$$x_{ij} \leftarrow x_{ij} + \delta \ , \quad g_i \leftarrow g_i - \delta \ , \quad g_j \leftarrow g_j + \delta \ .$$

If $g_i = 0$ and $x_{ij} < u_{ij}$, stop; else go to Step 1.

Step 3 (Decrease surplus of node i by reducing x_{ji} on incoming arcs).
Let $\delta = \min\{g_i, x_{ji} - l_{ji}\}$, update x_{ij}, g_i and g_j as follows:

$$x_{ji} \leftarrow x_{ji} - \delta \ , \quad g_i \leftarrow g_i - \delta \ , \quad g_j \leftarrow g_j + \delta \ .$$

If $g_i = 0$ and $x_{ji} > l_{ji}$, stop; else go to Step 1.

Step 4 (Increase price of node i). Set $\pi_i \leftarrow min_{\xi \in R_i^+ \cup R_i^-} \xi$, where

$$R_i^+ = \{\pi_j + c_{ij} + \epsilon \mid (i,j) \in \mathcal{E},\ x_{ij} < u_{ij}\}$$

$$R_i^- = \{\pi_j - c_{ji} + \epsilon \mid (j,i) \in \mathcal{E},\ x_{ji} > l_{ji}\}$$

If $R_i^+ \cup R_i^- = \emptyset$, the problem is infeasible; else go to Step 1.

For nodes with negative surplus, a symmetric down iteration can be defined. Our method for alternating between up and down iterations is described in Section 3.2.

2.1.1 Parallelizing the Flow Calculation Procedure. In order to obtain a highly parallelizable version of the algorithm we need to modify the sequential algorithm given above. For any node i, flows may be pushed out along all ϵ^+-unblocked outgoing arcs or ϵ^--unblocked incoming arcs. We want to push out flows simultaneously on all eligible arcs. However, the upper/lower bounds and the current flows of the arcs are different, leading to different "capacity of update" ($u_{ij} - x_{ij}$ or $x_{ij} - l_{ij}$) for each arc. Therefore we distribute the surplus/deficit g_i over the arcs in proportion to their capacities of update. Let

$$\begin{aligned} F_i &= \{j \mid (i,j) \text{ is } \epsilon^+ - \text{unblocked}\} \ , \\ B_i &= \{j \mid (j,i) \text{ is } \epsilon^- - \text{unblocked}\} \ , \\ \omega &= \max\{g_i, \sum_{j \in F_i} (u_{ij} - x_{ij}) + \sum_{j \in B_i} (x_{ji} - l_{ji})\} \ . \end{aligned}$$

Then for each arc $(i,j) \in \mathcal{E}$ with $j \in F_i$, x_{ij} is increased by the amount $g_i \frac{u_{ij} - x_{ij}}{\omega}$. For each arc $(j,i) \in \mathcal{E}$ with $j \in B_i$, x_{ji} is decreased by the amount $g_i \frac{x_{ji} - l_{ji}}{\omega}$. It

is easy to verify that the surplus computed with the updated flows is strictly less than the starting surplus g_i.

This approach was proposed by Li and Zenios [1991]. One transportation problem with 100 nodes and 900 arcs was solved using the sequential flow update and the parallel flow update procedure, respectively, on the CM–2. In the former case, the algorithm converges after 3450 iterations with 440.11 sec CM time, while in the latter case, it terminates after 1800 iterations and 13.25 sec CM time.

2.2 Proximal Minimization Algorithm with D-functions (PMD). The PMD algorithm was proposed in Censor and Zenios [1992], where its convergence was established. Let $S \neq \emptyset$ be an open convex set. Let $f : \Lambda \subseteq \Re^n \mapsto \Re$ be an auxiliary function. We assume that $\bar{S} \subseteq \Lambda$, where $\bar{S}$ is the closure of S, and that f is strictly convex and continuous on $\bar{S}$ and continuously differentiable on S. The set S is called the *zone* of f. The *D-function* corresponding to f is defined by

$$D_f(x, y) = f(x) - f(y) - \nabla f(y)^T (x - y). \tag{5}$$

Consider again the linear network flow problem [**LNFP**], which we restate as follows:

$$\min_{x \in X} c^T x, \tag{6}$$

where $X \subset \Re^n$ is the feasible region, $X = \{x \in \Re^n \mid Ax = b, l \leq x \leq u\}$, assumed to be nonempty, and A is the node-arc incidence matrix implied by (2). For some suitable choice of the auxiliary function f and a positive sequence $\{\gamma^k\}_{k=0}^{\infty}$ with $\liminf_{k \to \infty} \gamma^k = \gamma < \infty$, the **proximal minimization algorithm with D-functions** proceeds from an arbitrary starting point, $x^0 \in S$, with the following iteration.

The PMD Algorithm.

$$x^{k+1} \leftarrow \arg \min_{x \in X \cap \bar{S}} c^T x + \frac{1}{\gamma^k} D_f(x, x^k). \tag{7}$$

2.2.1 Quadratic Proximal Terms. Consider the PMD algorithm with the auxiliary function:

$$f(x) = \frac{1}{2} \| x \|^2, \tag{8}$$

where $\| \cdot \|$ denotes the Euclidean norm. The corresponding D-function defined by (5) is then $D_f(x, y) = \frac{1}{2} \| x - y \|^2$. In this case $\Lambda = S = \bar{S} = \Re^n$, and we obtain as a special case of (7) the **quadratic proximal point algorithm (QPP)** of Rockafellar [1976a, 1976b]:

The QPP Algorithm.

$$x^{k+1} \leftarrow \arg\min_{x \in X} c^T x + \frac{1}{2\gamma^k} \parallel x - x^k \parallel^2 . \tag{9}$$

For linear programs, QPP is finitely convergent. Also, there exists a $\bar{\gamma} < \infty$ such that for any $\gamma^k \geq \bar{\gamma}$ and any fixed x^k, the solution to the minimization in (9) is also a solution to (6) (see, e.g., Bertsekas and Tsitsiklis [1989, Section 3.4.3]).

2.2.2 Entropy Proximal Terms. To obtain another variant of the PMD algorithm, let $\text{ent} : \Re^n \mapsto \Re$ be the negative entropy function

$$\text{ent}(x) \doteq -\sum_{j=1}^{n} x_j \log\left(\frac{x_j}{a_j}\right), \tag{10}$$

where $a = (a_j) \in \Re^n$ is a given positive vector, log is the natural logarithm, and $0 \log 0$ is defined to be 0. By letting $f(x) = -\text{ent}(x)$ we obtain the D-function

$$D_f(x, y) = \sum_{j=1}^{n} x_j \left[\log\left(\frac{x_j}{y_j}\right) - 1\right] + \sum_{j=1}^{n} y_j . \tag{11}$$

f is defined on $\Lambda = \Re^n_+$, with zone $S = \{x = (x_j) \in \Re^n \mid x_j > 0\}$. It was shown to be a Bregman's function in, e.g., Censor et al. [1990]. Applying iterate (7) we obtain the **entropic proximal point algorithm (EPP):**

The EPP Algorithm.

$$x^{k+1} \leftarrow \arg\min_{x \in X \cap \bar{S}} c^T x + \frac{1}{\gamma^k} \left(\sum_{j=1}^{n} x_j \left[\log\left(\frac{x_j}{x_j^k}\right) - 1\right] + \sum_{j=1}^{n} x_j^k \right) \tag{12}$$

The constant term $\sum_{j=1}^{n} x_j^k$ can be removed from the optimization.

2.2.3 The Subproblem Algorithm. The PMD algorithm as described by the iteration (7) involves the solution of a series of strictly convex, separable subproblems. In the case of (9) we have a quadratic subproblem, and for (12) we have an entropy subproblem. The quadratic and entropic objective functions both belong to the class of *Bregman's functions*, as defined by Censor and Lent [1981], Hence, we can solve the subproblems using the primal-dual row-action algorithm of Censor and Lent [1981].

With the row-action algorithm, dual feasibility and complementary slackness are maintained throughout, but primal feasibility is obtained only in the limit. One dual variable is associated with each network node. Given dual values, the primal (flow) values satisfying CS are uniquely determined. The algorithm iteratively calculates, for each network node, the current surplus or deficit (4). Based on this value, and on the objective costs of the arcs incident to the nodes, the dual price of the node is updated. Simultaneously, the incident arc flows are updated, resulting in a reduced surplus or deficit. This combined update is

known as a *Bregman projection*. We implement the algorithm on the CM–2 as a Jacobi-type algorithm, whereby node prices are updated concurrently across the network. The algorithm and its specialization for the quadratic or entropic network problems, and its implementation on the CM–2 are described in detail in Nielsen and Zenios [1992c].

3. A Comparative Computational Study

In this section we compare the ϵ-relaxation and PMD algorithms to each other, and to a network simplex code running on a Cray Y–MP. We then investigate the question of scalability of these algorithms on the massively parallel architecture, and then compare a small-scale, parallel Gauss-Seidel implementation of the QPP algorithm to GENOS, a sequential network simplex. Finally, we compare the QPP and EPP algorithms on the CM–2 with a sequential implementation of Relax, Bertsekas [1991].

The two algorithms were implemented on a massively parallel Connection Machine CM–2 with up to 64K processors. The implementations were based on the data structures of Zenios and Lasken [1988]. Each network arc is assigned two processors, one at the head, the other at the tail of the arc. These processors handle the computation of flows which together with the current dual iterate satisfy CS, or ϵ-CS, depending on the algorithm. In addition, each network node is assigned a processor, which handles node supply or demand. The processors corresponding to a network node, the processors corresponding to the tails of arcs going out of the node, and to the heads of arcs coming into the node, are grouped together as a *segment*, allowing efficient computation (using *segmented scan* operations) of the surplus or deficit of each node. All processors in a node segment calculate the primal/dual update as a function of the node surplus/deficit and the objective coefficients of the incident arcs. The dual price updates are then communicated (using the general *send* operation) from the head to the tail of each arc, and vice versa. For details of the implementations, see Li and Zenios [1991] or Nielsen and Zenios [1992b]. Comparisons of alternative data structures — that concluded that the scheme outlined above is the most efficient — are given in Nielsen and Zenios [1992a].

3.1 Test Problems. The implementations of the algorithms discussed in Section 2 were tested on a series of problems arising from several large scale applications, and on a set of randomly generated transportation problems. TSENG1–TSENG8 are transportation problems generated by NETGEN of Klingman et al. [1974]. (See Zenios and Censor [1991] for a discussion of nonlinear variants of these models). The coefficients $\{c_{ij}\}$ are random numbers uniformly distributed in the interval [1,5], and supply/demand vectors are randomly generated in the interval [1, 1000]. The second class of problems was obtained from a Military Airlift Command application. They are referred to as the Patient Distribution System (PDS) problems and are used to make decisions on the evacuation of patients from Europe. PDSt denotes a problem that models a scenario of t days. They are linear multicommodity network problems with eleven commodities. We

Problem	Nodes	Arcs	Max. arc cost
TSENG1	500+500	5063	5
TSENG2	750+750	7611	5
TSENG3	1000+1000	10126	5
TSENG4	1250+1250	12665	5
TSENG5	500+500	10051	5
TSENG6	750+750	15086	5
TSENG7	1000+1000	20134	5
TSENG8	1250+1250	25153	5
PDS1.1	126	339	99999
PDS5.1	686	2149	99999
PDS10.1	1399	4433	99999
PDS20.1	2857	10116	99999
PDS30.1	4223	15526	99999
NAVY	3498	6841	1
HUGENAVY	30639	64542	0

Table 1: Medium scale test problem characteristics.

extracted the first commodity from each model to form our linear min-cost network problem. The size and complexity of the models increase with t. NAVY and HUGENAVY are two instances of a Naval personnel assignment model which was formulated as a very large linear program with an embedded network structure. We solve only the network component here. The problem NAVY has a nonzero assignment cost vector c whereas the larger problem HUGENAVY has an assignment cost vector which is identically zero. Pinar and Zenios [1992] describe these models in detail. The characteristics of these test problems are summarized in Table 1. Finally, two sets of very large problems were generated: the TPn problems were generated using NETGEN, while the GenNN problems

Problem	Nodes	Arcs	Max. arc cost	Average supply (per supply node)
TP00	5000	50000	100	100
TP01	5000	100000	100	200
TP1	10000	1000000	100	500
TP2	20000	1000000	100	1000
TP5	50000	1000000	100	400
TP10	50000	2000000	100	800
Gen0.25M	32768	245760	10	1500
Gen0.5M	65536	491520	10	1500
Gen16M	2097152	15728640	10	1500

Table 2: Large scale test problem characteristics.

were generated directly on the CM–2 due to their extremely large size, see Table 2. The parameters specified for TP1, TP2 and TP5 are identical to those used by Barr and Hickman [1990].

The termination criterion is small primal error, $\max_{i \in \mathcal{N}} \mid g_i \mid \leq \tau$, where g_i is calculated from (4) and τ is chosen as $\tau = 10^{-3} \frac{1}{m} \sum_{i=1}^{m} \mid b_i \mid$. The termination tolerance was checked every 25-50 iterations.

3.2 Testing the ϵ-relaxation algorithm. In this section we describe several implementation techniques that improve the convergence rate of the ϵ-relaxation algorithm, and then give results for the test problems described above.

The Initial Dual Point. The algorithm may be initialized by setting $\pi_i = 0$ for all $i \in \mathcal{N}$ and then setting $x_{ij} = l_{ij}$ or $x_{ij} = u_{ij}$ depending on the ϵ-CS conditions. This starting point works well for the bipartite transportation models, like TSENG1–8. However, it is a bad starting point if the network has a large diameter, with supply and demand nodes at the extremes.

A better initial dual point can easily be obtained by using a shortest path algorithm on the network graph. From the CS conditions for arc $(i,j) \in \mathcal{E}$, we have that if $l_{ij} < x_{ij} \leq u_{ij}$, then $\pi_i - \pi_j = c_{ij}$. If j is a destination node with $\pi_j = 0$, then π_i must be equal to c_{ij}. In general, if there is flow from a node i to a node j, then $\pi_i - \pi_j$ should be more than the length of the shortest path from i to j, where the "length" of arc (i,j) is c_{ij}. We hence implemented a variant of the Bellman-Ford shortest path algorithm and used it for providing an initial dual point.

This procedure is executed very efficiently on the CM. It is used only once at the beginning of the algorithm, and usually takes less than 1 sec of CM time. The initialization procedure is especially beneficial for the networks with large diameter, such as PDS20.1 and PDS30.1, where the overall solution time was reduced by a factor of two. The initialization routine is never detrimental, and is used henceforth.

ϵ-Scaling and Dynamic Tolerance Adjustment. The final value of ϵ needed for a precise solution is very small, and if this value is used throughout the algorithm, progress is extremely slow. This suggests the idea of *ϵ-scaling*, which was first analyzed for the linear network flow problem by Goldberg [1986], and was more fully established by Goldberg and Tarjan [1987]. Wein and Zenios [1991] incorporated ϵ-scaling in the auction algorithm for the solution of the assignment problem. ϵ-scaling consists of applying the algorithm in a series of phases, starting with a large value of ϵ and successively reducing ϵ up to an ultimate value which is less than the critical value $\frac{1}{m}$. Each application of the algorithm provides good initial prices for the next phase of application.

The sequence of values used for ϵ is given by:

$$\epsilon(k) = \max\{\frac{1}{m+1}, \frac{C}{m\theta^{k+1}}\}, \quad k = 0, 1, 2, \ldots.$$

where C is the maximum absolute cost coefficient, and $\theta > 1$ is a parameter set by the user; we used $4 \leq \theta \leq 8$. Experiments show that for problems with larger cost range it is beneficial to choose larger θ.

The second strategy is to introduce *adaptive tolerance* (also referred to as *node scaling*) technique. We give a larger error tolerance, τ, at the beginning of the iterations. During the course of the algorithm, we reduce τ by a factor of 2 in each phase of the ϵ-scaling, until we reach the desired tolerance. We tested a problem with 100 nodes and 900 arcs using the original algorithm, the ϵ-scaling variant and the adaptive tolerance variant. The scaled version terminated almost 20 times faster than the original algorithm, and the total number of iterations was reduced by a factor of about 20. When adding the adaptive tolerance procedure, we further reduced the solution time by 15%-20%.

Mixing Up and Down Iterations. We discussed in Section 2.1 the up iteration for positive surplus nodes. A symmetric iteration can be used for nodes with negative surplus (called *down iteration*). The down iteration only applies to nodes i with $g_i < 0$. Down iterations increase g_i and decrease p_i.

However, the algorithm may not terminate if up and down iterations are mixed arbitrarily. It is therefore necessary to impose some assumptions either on the problem structure or on the method by which up and down iterations are interleaved. Our model works as follows: We use two parameters to control the number of up iterations mixed with the number of down iterations. When we switch from up iteration to down iteration (or vice versa), we need to make sure that the dual objective value at this point does not deteriorate. Otherwise we will stay with our current form of iteration. In our experiments, about 10 up iterations were sufficient before switching to down iterations, and vice versa.

Tests showed that combining up and down iterations speeds up the convergence and is especially useful for transshipment problems. For example, the execution time of PDS1.1 is reduced from 10.28 sec to 8.71 sec, PDS5.1 from 23.55 sec to 16.92 sec and PDS10.1 from 40.56 sec to 26.42 sec. PDS20.1 and PDS30.1 did not converge after 10000 iterations without down iterations. After adding the down iterations PDS20.1 converges after 5600 iterations and PDS30.1 converges after 7300 iterations.

3.2.1 Results for the ϵ-Relaxation Algorithm. We now provide a summary of computational results in order to highlight certain aspects of the algorithm. Table 3 reports the results we have obtained from solving all test problems with up to 1 million arcs. During our experiments we found that PDS problems are relatively more difficult than the others. Two reasons explains this difficulty: one is that they have large cost (i.e., 99999) in the network. We can only avoid this difficulty by resorting to ϵ-scaling. Another reason is that large chains are present in the network. For instance, the diameter of the PDS30.1 network is 63. This observation motivated the use of a shortest path routine for finding a good dual starting point. HUGENAVY is a medium-sized problem. It is well scaled because all arcs have 0 costs. It took very few iterations to converge. This demonstrates that the ϵ-algorithm is much faster on problems with small arc costs. As a final exercise, we solved the large-scale problems: TP00, TP01, TP1, TP2 and TP5. These are very sparse transportation problems (0.8%, 1.6%, 4%, 1% and 0.16% sparse, respectively). They have medium arc costs [1,100], which does not cause much difficulty.

Problem	Time	Itns.	VP ratio
TSENG1	32.80	2650	1
TSENG2	30.04	2200	1
TSENG3	43.15	3100	1
TSENG4	47.30	3350	1
TSENG5	37.25	2750	1
TSENG6	39.60	2800	1
TSENG7	55.27	2900	2
TSENG8	44.56	2350	2
PDS1.1	8.71	500	1
PDS5.1	16.92	950	1
PDS10.1	26.42	1900	1
PDS20.1	103.25	5600	1
PDS30.1	135.90	7300	2
NAVY	0.92	50	1
HUGENAVY	2.47	50	4
TP00	88.43	3250	4
TP01	164.84	3050	8
TP1	760.45	2050	64
TP2	956.26	2950	64
TP5	890.76	2700	64

Table 3: Results for the ϵ-relaxation algorithm.

3.3 Testing the Proximal Minimization Algorithms. By *minor iteration* we mean a complete sweep of the row-action algorithm over all nodes of the network. Of course, in the massively parallel implementation this is done concurrently. A *major iteration* is the execution of a number of minor iterations followed by an update of the proximal point.

The *perturbation gap* δ_1 measures the difference between the objective value of the nonlinearly perturbed program, and the linear objective value. The final solution is denoted by x^k. The perturbation gap for QPP is $\delta_1 = \frac{1}{2\gamma^k} \parallel x^k - x^{k-1} \parallel^2$, and for EPP,

$$\delta_1 = \frac{1}{\gamma^k} \left(\sum_{j=1}^{n} x_j \left[\log \left(\frac{x_j^k}{x_j^{k-1}} \right) - 1 \right] + \sum_{j=1}^{n} x_j^{k-1} \right).$$

The *primal error* δ_2 measures the maximum absolute surplus or deficit at any network node, $\delta_2 = \max_{i \in \mathcal{N}} \{ \mid g_i \mid \}$. Besides these statistics, we also report the duality gap $b^T \pi - c^T x$ (i.e., the difference between the primal and dual objective values) at termination. Solution times are in seconds, on a dedicated 16K processor CM–2 using a SUN 4/360 front-end. The CM utilization factor for all runs is above 95%.

The algorithm terminates when the following criterion is satisfied: $\delta_1 \leq \eta_1$ and $\delta_2 \leq \eta_2$ where η_1 and η_2 are user specified parameters.

Problem	γ^0	Minor Iter.	Time	Obj. val.	Duality gap
Tseng1	0.2	3050	18.80	$5.792 \cdot 10^7$	-0.001%
Tseng2	0.2	5400	45.94	$8.798 \cdot 10^7$	0.003%
Tseng3	0.2	4025	32.49	$1.240 \cdot 10^8$	-0.001%
Tseng4	0.2	2750	22.76	$1.456 \cdot 10^8$	-0.002%
Tseng5	0.2	3350	28.50	$3.443 \cdot 10^7$	0.0002%
Tseng6	0.2	4025	35.53	$5.364 \cdot 10^7$	0.0004%
Tseng7	0.2	3750	47.23	$7.732 \cdot 10^7$	0.0005%
Tseng8	0.2	3700	48.54	$9.166 \cdot 10^7$	-0.0016%
PDS1.1	1.0	250	1.43	$2.655 \cdot 10^9$	-0.00002%
PDS5.1	1.0	550	3.73	$2.515 \cdot 10^9$	0.00003%
PDS10.1	1.0	950	5.67	$2.341 \cdot 10^9$	0.0001%
PDS20.1	1.0	2000	15.06	$1.978 \cdot 10^9$	0.0003%
PDS30.1	1.0	3850	45.19	$1.632 \cdot 10^9$	0.0007%
Navy	1000	1450	10.80	0	0.0002
Hugenavy	1000	1225	55.66	0	0.0031
TP00	1.0	2025	57.91	$2.901 \cdot 10^6$	0.0002%
TP01	1.0	4525	232.84	$3.169 \cdot 10^6$	0.0001%

Table 4: Solving the test problems using the Quadratic Proximal Point algorithm on a CM–2 with 16K processors.

The theory for QPP (Rockafellar [1976]) assures convergence if the kth subproblem is solved to an accuracy of $\delta_2^k \leq \epsilon^k$, where $\sum_{k=1}^{\infty} \epsilon^k < \infty$. No corresponding result is available for the EPP algorithm. We explored in Nielsen and Zenios [1992c] various strategies regarding subproblem solution accuracy, and found the simple strategy of executing 25-50 minor iterations for each major iteration (proximal point update) to be most efficient. We use the same strategy in this study. The convergence criteria are checked once every major iteration.

For the quadratic and entropic runs of this section, the penalty parameter γ^k and the proximal point were updated as follows:

$$\gamma^{k+1} = \begin{cases} \gamma^k & \text{if } \delta_2 > \eta_2, \\ 2\gamma^k & \text{if } \delta_2 \leq \eta_2, \end{cases} \tag{13}$$

Correspondingly, the proximal point was updated at each major iteration k as follows:

$$x^{k+1} = \begin{cases} \rho x - (1-\rho)x^k & \text{if } \delta_1 > \eta_1 \\ x^k & \text{if } \delta_1 \leq \eta_1, \end{cases} \tag{14}$$

where x is the current iterate of the subproblem, and $\rho \in (0,2)$ is a relaxation parameter. The proximal point is thus not updated if the perturbation error is within the tolerance.

3.3.1 Results for the PMD Algorithms. In Tables 4 and 5 we give detailed results of solving the medium sized test problems with the quadratic and entropic proximal point methods, respectively. A comparison is shown in Figure 1. All

Problem	γ^0	Minor Iter.	Time	Obj. val.	Duality gap
Tseng1	0.002	1900	19.34	$5.792 \cdot 10^7$	-0.0006%
Tseng2	0.002	1725	22.63	$8.797 \cdot 10^7$	-0.0005%
Tseng3	0.002	1775	22.61	$1.240 \cdot 10^8$	0.001%
Tseng4	0.002	2650	34.05	$1.455 \cdot 10^8$	-0.00006
Tseng5	0.002	2450	31.48	$3.443 \cdot 10^7$	0%
Tseng6	0.001	5250	71.87	$5.364 \cdot 10^7$	-0.0004%
Tseng7	0.002	2250	43.14	$7.733 \cdot 10^7$	-0.0002%
Tseng8	0.002	4375	83.79	$9.163 \cdot 10^7$	-0.0008%
PDS1.1	10^{-4}	400	3.77	$2.655 \cdot 10^9$	0.0009%
PDS5.1	10^{-4}	875	7.22	$2.515 \cdot 10^9$	0.003%
PDS10.1	10^{-4}	1400	14.04	$2.341 \cdot 10^9$	-0.002%
PDS20.1	10^{-4}	2625	33.53	$1.979 \cdot 10^9$	0.002%
PDS30.1	10^{-4}	3925	74.10	$1.634 \cdot 10^9$	-0.0027%
Navy	100	100	1.26	0.000023	-0.000020
Hugenavy	100	150	10.08	0	-0.00024
TP00	1/6	1700	67.41	$2.900 \cdot 10^6$	-0.0004%
TP01	1/6	1625	116.48	$3.165 \cdot 10^6$	0.0021%

Table 5: Solving the test problems using the Entropy Proximal Point algorithm on a CM–2 with 16K processors.

problems were solved to a perturbation tolerance of $\eta_1 = 10^{-5}$ and a primal tolerance of $\eta_2 = 0.5$.

Both algorithms solved all of the test problems in a reasonable time. For the QPP algorithm, the solution times were not very sensitive to the choice of initial penalty parameter γ^0, except for the Navy and Hugenavy problems: For a smaller value than the one shown, the convergence of the proximal point would be very slow. The EPP algorithm required more experimentation with the penalty parameter to obtain good convergence. For both algorithms, the larger the penalty parameter, the faster the proximal point sequence $\{x^k\}$ converged, but the subproblems became more difficult for the dual relaxation algorithm.

The duality gap at the solution point was for the Tseng and Navy problems generally much smaller with the EPP algorithm than with the QPP algorithm, whereas the opposite is true for the PDS problems. It is not clear which characteristics of the test problems account for this consistent difference.

The large-scale problems were solved to a primal error tolerance of 0.5, but (in contrast with the medium-size problems) to a perturbation tolerance of 10^{-4} to keep running times reasonable.

Both the QPP and the EPP algorithms solved the problems in a reasonable time. The results are given in Table 6. The EPP algorithm is uniformly faster than the QPP algorithm for these problems (but see Section 3.4). The main difference between the two algorithms was the tail convergence of the subproblem primal error: The EPP algorithm consistently reduced this error faster than the QPP algorithm.

Figure 1: Comparing the Quadratic and Entropic Proximal Point algorithms.

Problem	**Algorithm**	γ^0	**Minor Iter.**	**Time**
TP1	QPP	0.5	9000	2850.18
	EPP	1/12	1550	726.42
TP2	QPP	0.5	5300	1669.73
	EPP	1/12	1275	597.24
TP5	QPP	0.5	3200	1018.32
	EPP	1/24	1300	613.34
TP10	QPP	0.5	4875	3092.86
	EPP	1/24	1725	1616.27

Table 6: Solving large-scale test problems using the proximal point algorithms on a 16K CM–2.

Problem	QPP			EPP		
	ϵ^0	Iter.	Time	ϵ^0	Iter.	Time
Gen0.25M	10	2325	4.6 min.	0.01	9900	25.4 min.
Gen0.5M	10	2400	7.2 min.	0.01	8350	44.9 min.
Gen16M	1	3075	99 min.		N/A	N/A

Table 7: Solving the large-scale, random transportation problems. The smaller runs were executed on a 16K PE CM–2, the Gen16M run on a 32K PE CM–2 at a VP-ratio of 1024.

3.4 Solving extremely large-scale problems. In order to test the PMD algorithms on extremely large, linear network problems, a set of test problems were randomly generated directly on the CM–2. They have an equal number of supply- and demand-nodes, each with a degree of 15. The problem characteristics are given in Table 2. These problems were solved to a perturbation tolerance of 10^{-5} and a primal feasibility tolerance of 0.1.

Results are given in Table 7. It is interesting to note that these problems appear to be easier for the QPP algorithm than the TP problems. The largest problem, consisting of more than 2 million nodes and almost 16 million arcs, was solved using the QPP algorithm in about 1.5 hours on a 32K PE CM–2.

3.5 Scalability of the CM–2 Implementations. In order to see how well the implementations utilize the underlying parallel hardware, we solved a set of test problems of varying size using 8K, 16K and 32K processors. These problems were randomly generated using size as a parameter. Results are shown in Figure 2, where problem size is measured by the number of VPs required for the representation, $2n+m$ (the graph for 64K processors is estimated). We note that, for each machine size, as the problem size is increased by a factor of two, the solution time increases by about 1.8–1.9. Also, as the machine size is doubled, the solution time decreases by about the same factor, 1.8–1.9. That these factors are less than two is evidence that the CM–2 is used more efficiently with higher VP-ratios. However, the factors are close to two, which shows that our implementations scale efficiently with respect to both the machine and problem size.

3.6 Comparison with small-scale parallel algorithm. We now compare a Gauss-Seidel version of the QPP algorithm, running on a 8 processor Alliant FX/8, with the sequential GENOS network simplex. The test problems used are again transportation problems generated using NETGEN, with coefficients generated uniformly in the interval [1,1000]. Total demands and supplies are both $250 \cdot m$, where m is the total number of nodes, randomly allocated to the supply and demand nodes, respectively.

The results of this comparison, shown in Table 8, indicate that even a Gauss-Seidel version of QPP — which is superior to the Jacobi version since it uses at each node iteration data which are up-to-date — is not competitive with a state-

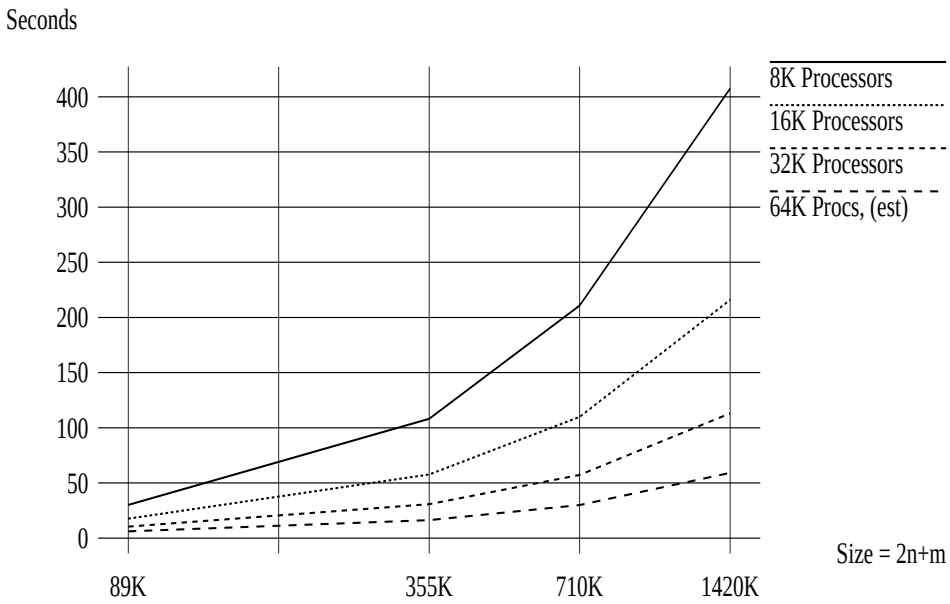

Figure 2: Solving problems of differing size using 8K, 16K and 32K CM–2 processors. Results for the 64K CM–2 are estimated.

Problem	Nodes	Arcs	QPP time	Simplex time
SMTRANS.00	35 + 45	185	18.17	0.343
LTRANS0.00	500 + 500	10046	35.6*	22.47
LTRANS1.00	1250 + 1250	12674	61.3*	68.21
LTRANS2.00	1250 + 1250	25152	112.5*	96.63

Table 8: Comparing a small-scale parallel Gauss-Seidel implementation of QPP to a network simplex. The runs marked * are parallelized to use 8 processors on the Alliant FX/8, the remaining times are using 1 FX/8 processor.

Problem	QPP time	Relax time	Ratio
PDS1.1	1.43	0.012	119
PDS5.1	3.73	0.090	41.4
PDS10.1	5.67	0.164	34.6
PDS20.1	15.06	0.504	29.9
PDS30.1	45.19	1.090	41.5
Navy	10.80	0.125	80.6
Hugenavy	55.66	2.465	22.6

Table 9: Comparing the QPP algorithm to the Relax algorithm. Times are in seconds: QPP on a 16K CM–2 versus the Relax running on a DECStation 5000.

of-the art simplex code when both are executing on serial, or small-scale parallel machines. However, the PMD algorithms are competitive on the massively parallel architectures, and only the PMD algorithms could solve the multi-million variable problems of Table 2. Due to this result, we did not further pursue sequential or small-scale, parallel implementations of the PMD algorithms.

3.7 Comparisons with Relax. We compare the QPP algorithm with the relaxation code Relax of Bertsekas [1991]. Relax was executed on a DECStation 5000 using those of our test problems which are pure networks with integer data. Results, given in Table 9, show that Relax is superior to the PMD algorithms for problems of this size. However, due to memory limitations we could not solve multi-million variable problems with the sequential implementation of Relax.

4. Comparison of the ϵ-relaxation and PMD algorithms

We now compare the ϵ-relaxation algorithm and the two PMD algorithms, based on the results of the preceding sections. The results for the ϵ-relaxation algorithm (Table 3) were obtained on a CM–2 with 32K processors, whereas the results for the PMD algorithms (Tables 4, 5 and 6) are from a 16K CM–2.

The ϵ-relaxation algorithm is very fast on the Navy and Hugenavy problems, and converges much faster than either of the PMD algorithms. These problems are characterized by a very small range of the objective coefficients, which benefits this algorithm. The fastest algorithm overall, however, is the EPP algorithm. On the medium sized problems it is comparable with the QPP algorithm, and they are both roughly twice as fast as the ϵ-relaxation algorithm (when scaling execution times to comparable machine sizes). For the large and extremely large problems, it is not clear whether the QPP or EPP algorithm is superior. This seems to depend on the individual problem characteristics. However, both these algorithms seem to clearly outperform the ϵ-relaxation algorithm, except for problems with very small objective coefficients.

Finally, we provide a comparison with a state-of-the-art network simplex code executing on a CRAY Y–MP, namely GENOS (Mulvey and Zenios [1987]). All CM–2 execution times are estimated for a 64K CM–2, using factors estimated

Figure 3: Comparing the EPP, ϵ-relaxation and network simplex algorithms. The figure includes results for the large-scale problems TP1 and TP5. EPP was run on a 16k CM–2, ϵ-relaxation on a 32k CM–2 and network simplex on a CRAY Y–MP.

from Figure 2. Due to the different nature of the hardware, algorithms and solution quality, one must be careful making conclusive claims, based upon solution time alone. With this in mind, we note that the proximal point algorithms, executing on the CM–2, are, with the exception of Hugenavy, not convincingly faster than the simplex algorithm executing on the CRAY Y–MP. However, the proximal point algorithms do appear to become increasingly competitive as the problem size increases. The largest TP or Gen problems could not be solved on the CRAY due to memory limitations.

5. Conclusions

Based on the results of this paper, we can answer affirmatively the question whether massively parallel algorithms can solve large linear network problems. Each of the algorithms investigated solve very large linear network problems efficiently. Generally, the Proximal Minimization Algorithms with D-functions are superior to the ϵ-relaxation algorithm, but much depends on the precise problem characteristics, such as the range of the objective coefficients.

All the algorithms have very similar memory requirements, and memory more than execution time was the limiting factor regarding the size of problems we could solve. It seems reasonable to conclude that even larger problems could be solved given access to a larger CM–2. On a CM–2 with 64K processors, each with 256Kbytes of memory, for instance, we would expect to be able to solve problems with more than 32 million variables.

REFERENCES

1. R.S. Barr and B.L. Hickman. A new parallel network simplex algorithm and implementation for large time-critical problems. Technical report 90–CSE–37, Southern Methodist University, Department of Computer Science and Engineering, 1990.

2. D.P. Bertsekas. A Distributed Algorithm for the Assignment Problem. Laboratory for Information and Decision Systems, Working Paper, Massachusetts Institute of Technology, Cambridge, 1979.

3. D.P. Bertsekas. Distributed asynchronous relaxation methods for linear network flow problems. Report LIDS-P-1606, MIT, Nov. 1986a.

4. D.P. Bertsekas. Distributed relaxation methods for linear network flow problems. *Proc. 25th IEEE Conf. Dec. & Contr.*, pages 2101–2106, 1986b.

5. D.P. Bertsekas. Linear Network Optimization, The MIT Press, Cambridge, Massachusetts, 1991.

6. D.P. Bertsekas and J.N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods.* Prentice Hall, Englewood Cliffs, New Jersey, 1989.

7. Y. Censor and A. Lent. An iterative row-action method for interval convex programming. *Journal of Optimization Theory and Applications*, 34:321–353, 1981.

8. Y. Censor, A.R. De Pierro, T. Elfving, G.T. Herman, and A.N. Iusem. On iterative methods for linearly constrained entropy maximization. In A. Wakulicz, editor, *Numerical Analysis and Mathematical Modelling*, volume 24, pages 145–163. Banach Center Publications, PWN - Polish Scientific Publisher, Warsaw, Poland, 1990.

9. Y. Censor and S.A. Zenios. The proximal minimization algorithm with D-functions. *Journal of Optimization Theory and Applications*, 73(3):455–468, 1992.

10. J.R. Eriksson. An iterative primal-dual algorithm for linear programming. Report LiTH-MAT-R-1985-10, Department of Mathematics, Linkøping University, S-581 83 Linkøping, Sweden, October 1985.

11. A. V. Goldberg. Solving minimum cost flow problems by successive combinatorial algorithms. In *Proc. 18th. ACM STOC.*, pages 136–146, 1986. Extented Abstract.

12. A.V. Goldberg and R.R. Tarjan. Solving minimum cost flow problems by successive approximation. In *Proc. 10th. ACM STOC.*, pages 7–18, 1987.

13. D. Klingman, A. Napier, and J. Stutz. NETGEN - a program for generating large-scale (un)capacitated assignment, transportation, and minimum cost flow network problems. *Management Science*, 20:814–822, 1974.

14. X. Li and S. A. Zenios. Data-level parallel solution of min-cost network flow problems using ϵ-relaxation. Report 91–05–04, Decision Sciences Department, The Wharton School, University of Pennsylvania, Philadelphia, PA 19104, May. 1991.

15. M.Ç.Pinar and S.A. Zenios. Naval personnel assignment: An application of linear-quadratic penalty methods. In O. Balci, R. Sharda and S.A. Zenios., eds, *Computer Science and Operations Research: New developments in their Interfaces*, Pergamon Press, pp. 43–58, 1992.

16. J.M. Mulvey and S.A. Zenios. GENOS 1.0: A Generalized Network Optimization System. User's Guide. Report 87–12–03, Decision Sciences Department, the Wharton School, University of Pennsylvania, Philadelphia, PA 19104, 1987.

17. S.S. Nielsen and S.A. Zenios. Data Structures for Network Algorithms on Massively Parallel Architectures, *Parallel Computing* 18:1033–1052, 1992a.

18. S.S. Nielsen and S.A. Zenios. Massively parallel algorithms for singly constrained nonlinear programs. *ORSA Journal on Computing*, 4(2):166–181, 1992b.

19. S.S. Nielsen and S.A. Zenios. Proximal minimization with D-functions and massively parallel solution of linear network programs. *Computational Optimization and Applications*, 1(4), 1992c (to appear).

20. R.T. Rockafellar. Augmented Lagrangians and applications to proximal point algorithms in convex programming. *Math. of Operations Research*, 1:97–116, 1976.

21. R.T. Rockafellar. Monotone operators and the proximal point algorithm. *SIAM Journal on Control and Optimization*, 14:877–898, 1976.

22. J. Wein and S.A. Zenios. On the massively parallel solution of the assignment problem. *Journal of Parallel and Distributed Computing*, 13:228–236, 1991.

23. S.A. Zenios and Y. Censor. Massively parallel row-action algorithms for some nonlinear transportation problems. *SIAM Journal of Optimization*, 1:373–400, 1991.

24. S.A. Zenios and R.A. Lasken. Nonlinear network optimization on a massively parallel Connection Machine. *Annals of Operations Research*, 14:147–165, 1988.

Soren S. Nielsen, Management Science and Information Systems, University of Texas, Austin, TX 78729. *E-mail:* nielsen@bongo.cc.utexas.edu

Stavros A. Zenios, Wharton School of Business, University of Pennsylvania, Philadelphia, PA 19104. *E-mail:* zenios@wharton.upenn.edu