ON THE FINE-GRAIN DECOMPOSITION OF MULTI-COMMODITY TRANSPORTATION PROBLEMS

by

Stavros A. Zenios

OR 236-90 November 1990

On the Fine-Grain Decomposition of Multicommodity Transportation Problems

Stavros A. Zenios

Decision Sciences Department
The Wharton School
University of Pennsylvania
Philadelphia, PA 19104.

October, 1990.

Abstract

We develop algorithms for nonlinear problems with multicommodity transportation constraints. The algorithms are of the row-action type and, when properly applied, decompose the underlying graph alternatingly by nodes and edges. Hence, a fine-grain decomposition scheme is developed that is suitable for massively parallel computer architectures of the SIMD (i.e., single instruction stream, multiple data stream) class.

Implementations on the Connection Machine CM–2 are discussed for both dense and sparse transportation problems. The dense implementation achieves computing rate of 1.6–3 GFLOPS. Several aspects of the algorithm are investigated empirically. Computational results are reported for the solution of quadratic programs with approximately 10 million columns and 100 thousand rows.

Contents

1 Introduction

Data-level parallelism appears to be a successful paradigm for computing on massively parallel architectures. It is based on the premise that a massively parallel algorithm should use multiple processors to carry out identical operations on different parts of the input data. Communication among processors is orderly and synchronous. With this approach one avoids the difficulties encountered in coordinating thousands — or, potentially, millions — of processors in an asynchronous, chaotic fashion. Even more importantly, however, is the ability to develop abstract models for data-level parallel computing: the vector random-access-machine (V-RAM) of Blelloch [1990]. It is thus possible to study complexity issues of massively parallel algorithms, and hence gain insight in their efficiency in an abstract setting and without actual implementations. A potential limitation of data-level parallelism is the requirement that the operations of the algorithm are identical on all the data. To what extend this mode of computing can be applied to solve a broad range of problems is presently unclear.

The focus of this paper is the design of fine-grain decomposition algorithms for nonlinear optimization problems with multicommodity transportation constraints. We seek algorithms that decompose the problem into a large number of independent and identical subproblems, and are, therefore, suitable for data-level parallel computing. If a problem requires $O(L)$ steps, a fine-grain decomposition will require $L \cdot O(1)$ steps. On a computer system with P processors the problem can be solved in $\lceil \frac{L}{P} \rceil \cdot O(1)$ steps. When P is large enough — as is the case with massively parallel computers — the problem can be solved in a constant number of operations which is independent of its size.

Fine-grain decomposition algorithms are designed here for a class of multicommodity transportation problems. Such problems appear in operations research (logistics, distribution, manufacturing, etc.), computer science (communication routing) and transportation (the problem of estimating origin/destination tables, which is similar to the estimation of social accounting matrices in development planning and the estimation of migration patterns in regional sciences). This is the second in a series of three papers that develop massively parallel algorithms for optimization problems with network structures: The single commodity transportation problem was studied in Zenios and Censor [1990] and algorithms for stochastic network problems are the topic of the paper by Nielsen and Zenios [1990a].

Linear multicommodity network flow problems have received extensive investigation; see the survey articles by Assad [1978] or Kennington [1978]. Very little has been done on the nonlinear problem, and in the cases known to the author it has been motivated by recent developments in parallel computing: Shultz and Meyer [1989] and Zenios, Pinar and

Dembo [1990]. The (easier) problem when the multiple commodities jointly contribute in a congestion function has been studied in greater length: Bertsekas [1979] and Gallager [1977] for data communication networks, and Chen and Meyer [1988] for traffic assignment problems. The problems considered here have a more general constraint set than these earlier studies: they permit multipliers (i.e., gains) on individual commodities, and weighted linear combinations of the commodities contribute to the coupling constraints. This is the generalized multicommodity flow problem. The only study that dealt with this problem is Wollmer's [1972].

The algorithmic approach we follow is based on the class of row-action algorithms of Censor [1981]. These algorithms have been proven successful in the solution of very large, sparse optimization problems that arise in medical imaging, Herman [1980]. With the appropriate choice of some control parameters, the algorithms can be implemented in parallel: for a classification see Censor [1988], and for applications Zenios and Censor [1988] and Censor and Zenios [1989a]. A limitation of this approach is that the developed algorithms are specialized for quadratic or entropy objective functions. Such functions are typically used in the matrix estimation applications mentioned above. However, these algorithms can not solve directly the linear programming case. They are, nevertheless, the building blocks for more general algorithmic schemes that can be applied to linear programs: the proximal minimization algorithm of Rockafellar [1976], or the proximal minimization algorithm with D-functions of Censor and Zenios [1989b].

One of the developed algorithms is been implemented on a massively parallel Connection Machine CM–2 with up to 32K processing elements. We develop the data structures for the representation of sparse, multicommodity network problems. To this end we extend the data structures of Zenios and Lasken [1988] to represent multiple single-commodity networks. The implementation is used to study the performance of the algorithm. Numerical results are analyzed for the solution of test problems with up to 10 million variables and 100 thousand constraints.

We define now the problem and establish notation.

1.1 Problem Formulation

We will use $< m >$ to denote the set $\{1, 2, 3, ..., m\}$. $\Re^m$ is the m-dimensional Euclidean space and $< \cdot, \cdot >$ is the Euclidean inner product. $mid\{\alpha, \beta, \gamma\}$ is the median of the the three real numbers, α, β and γ. If $\alpha \leq \gamma$ the median can be computed as $\max\{\alpha, \min\{\beta, \gamma\}\}$. We use $\lceil m \rceil_2$ to denote rounding up of m to the next integer that is a power of 2 (e.g., $\lceil 65 \rceil_2 = 128$).

A transportation graph is defined by the triplet $G = (V_O, V_D, \mathcal{E})$ where $V_O =< m_O >$, $V_D =< m_D >$ and $\mathcal{E} \subseteq \{(i,j) | i \in V_O, j \in V_D\}$. V_O and V_D are the sets of origin and destination nodes with cardinality m_O and m_D respectively. $\mathcal{E}$ is the set of n directed arcs (i,j) with origin i and destination j which belong to the graph, $(n \leq m_O\, m_D)$. On this transportation problem we consider the flow of K distinct commodities and let $< K >$ denote the set of these commodities. We assume for symmetry of notation that the underlying graph is identical for all commodities. Additional notation is needed to define the transportation problem for each commodity, and then to define the joint restrictions over all the commodities.

For each commodity $k \in< K >$ we have:

$x_k = (x_k(i,j)) \in \Re^n$, $(i,j) \in \mathcal{E}$, is the vector of flows,

$u_k = (u_k(i,j)) \in \Re^n$, $(i,j) \in \mathcal{E}$, is the vector of upper bounds on the flows,

$m_k = (m_k(i,j)) \in \Re^n$, $(i,j) \in \mathcal{E}$, is the vector of multipliers on the flows,

$s_k = (s_k(i)) \in \Re^{m_O}$, $i \in V_O$, is the vector of supplies,

$d_k = (d_k(j)) \in \Re^{m_D}$, $j \in V_D$, is the vector of demands,

$\pi_k^O = (\pi_k^O(i)) \in \Re^{m_O}$, $i \in V_O$, is the vector of dual prices for the origin nodes,

$\pi_k^D = (\pi_k^D(j)) \in \Re^{m_D}$, $j \in V_D$, is the vector of dual prices for the destination nodes,

$r_k = (r_k(i,j)) \in \Re^n$, $(i,j) \in \mathcal{E}$, is the vector of dual prices for the bound constraints.

For the joint capacity constraints we define:

$U = (U(i,j)) \in \Re^n$, $(i,j) \in \mathcal{E}$, the vector of mutual arc capacities,

$e_k = (e_k(i,j)) \in \Re^n$, $(i,j) \in \mathcal{E}, k \in< K >$, coefficients indicating the contribution of commodity k to the mutual capacity of arc (i,j),

$\psi = (\psi(i,j)) \in \Re^n$, $(i,j) \in \mathcal{E}$, the vector of dual prices for the joint capacity constraints.

In order to define the node-arc incidence relationships — assumed to be identical for all commodities — we define:

$\delta^+(i) = \{j \in V_D | (i,j) \in \mathcal{E}\}$, the set of destination nodes which have incident arcs with origin node i,

$\delta^-(j) = \{i \in V_O | (i,j) \in \mathcal{E}\}$, the set of origin nodes which have incident arcs with destination node j.

We define the pure and generalized multicommodity transportation problems as follows:

[PMTR] Pure Multicommodity Transportation Problem

$$\begin{aligned}
&\textit{Minimize} \quad F(x) && && (1)\\
&\textit{Subject to :} && && \\
&\sum_{j\in\delta^+(i)} x_k(i,j) &=\ & s_k(i),\ \forall i \in V_O,\ k \in < K > && (2)\\
&\sum_{i\in\delta^-(j)} x_k(i,j) &=\ & d_k(j),\ \forall j \in V_D,\ k \in < K > && (3)\\
&0 \le x_k(i,j) &\le\ & u_k(i,j),\ \forall (i,j) \in \mathcal{E},\ k \in < K > && (4)\\
&\sum_{k\in<K>} x_k(i,j) &\le\ & U(i,j),\ \forall (i,j) \in \mathcal{E} && (5)
\end{aligned}$$

[GMTR] Generalized Multicommodity Transportation Problem

$$\begin{aligned}
&\textit{Minimize} \quad F(x) && && (6)\\
&\textit{Subject to :} && && \\
&\sum_{j\in\delta^+(i)} x_k(i,j) &\le\ & s_k(i),\ \forall i \in V_O,\ k \in < K > && (7)\\
&\sum_{i\in\delta^-(j)} m_k(i,j)\cdot x_k(i,j) &=\ & d_k(j),\ \forall j \in V_D,\ k \in < K > && (8)\\
&0 \le x_k(i,j) &\le\ & u_k(i,j),\ \forall (i,j) \in \mathcal{E},\ k \in < K > && (9)\\
&\sum_{k\in<K>} e_k(i,j)\cdot x_k(i,j) &\le\ & U(i,j),\ \forall (i,j) \in \mathcal{E} && (10)
\end{aligned}$$

The objective function $F : \Re^{nK} \to \Re$ may take one of the following two forms:

[Q] Quadratic

$$F(x) = \sum_{k\in<K>} \sum_{(i,j)\in\mathcal{E}} \frac{1}{2} w_k(i,j)\cdot x_k^2(i,j) + c_k(i,j)\cdot x_k(i,j) \tag{11}$$

where $\{w_k(i,j)\}$ and $\{c_k(i,j)\}$ are given positive real numbers.

[E] Entropy

$$F(x) = \sum_{k\in<K>} \sum_{(i,j)\in\mathcal{E}} x_k(i,j)\cdot\left[\ln\left(\frac{x_k(i,j)}{a_k(i,j)}\right) - 1\right] \tag{12}$$

where ln is the natural logarithm and $\{a_k(i,j)\}$ are given positive real numbers.

This discussion concludes the formulation of the problems for which we develop algorithms in the sequel. The following assumptions are made for both [PTMR] and [GMTR].

Assumption 1: The feasible sets defined by (2)–(5) and (7)–(10) respectively are non-empty. The existence of feasible solutions for multicommodity network flow constraints can be checked using the algorithm in Gondran and Minoux [1986, pp. 251–256].

Assumption 2: The transportation graph G is connected. Otherwise, the problem could be partitioned into its disconnected components and each solved separately.

Assumption 3: $s_k(i) > 0$, $\forall i \in V_O$, $k \in< K >$ and $d_k(j) > 0\ \forall j \in V_D$, $k \in< K >$. If these conditions are violated for some index i or j for some commodity k, then all the flows for the particular commodity on arcs incident to the offending node can be set to zero, and the relevant constraint removed from the problem.

1.2 Matrix Formulations

In order to develop the row-action algorithms we will need to express the problems in a compact matrix notation. Constraints (2)–(5) are rewritten as:

$$Sx = s \tag{13}$$

$$Dx = d \tag{14}$$

$$0 \leq Ix \leq u \tag{15}$$

$$0 \leq Ex \leq U \tag{16}$$

I is the $nK \times nK$ identity matrix. $x \in \Re^{nK}$ is the vector $(x_1 \mid x_2 \mid \ldots \mid x_K)^T$, $u \in \Re^{nK}$ is the vector $(u_1 \mid u_2 \mid \ldots \mid u_K)^T$, $s \in \Re^{m_O K}$ is the vector $(s_1 \mid s_2 \mid \ldots \mid s_K)^T$ and $d \in \Re^{m_D K}$ is the vector $(d_1 \mid d_2 \mid \ldots \mid d_K)^T$.

S is the $m_O K \times n$ matrix:

$$S = \begin{bmatrix} \alpha_{11}^1 \ldots \alpha_{1m_D}^1 & & & | & & & \\ & \ldots & & | & & & \\ & & \alpha_{m_O1}^1 \ldots \alpha_{m_O m_D}^1 & | & & & \\ \text{------} & \text{--} & \text{--------} & & & & \\ & & & \ldots & & & \\ & & & & \text{------} & \text{--} & \text{--------} \\ & & & & \mid \alpha_{11}^K \ldots \alpha_{1m_D}^K & & \\ & & & & \mid & \ldots & \\ & & & & \mid & & \alpha_{m_O1}^K \ldots \alpha_{m_O m_D}^K \end{bmatrix}$$

with entries a_{ij}^k, for all $k \in< K >$, given by

$$\alpha_{ij}^k = \begin{cases} 1, & \text{if } j \in \delta^+(i), \\ 0, & \text{otherwise.} \end{cases} \tag{17}$$

The remaining entries of S are all zeros. D is the $m_D K \times n$ matrix

$$D = \begin{bmatrix} \beta_{11} & & & \beta_{m_O 1} & & \\ & & \cdots & & & \\ & \beta^1_{1 m_D} & & & \beta^1_{m_O m_D} & \\ \hline & & & & & \cdots \\ \hline \beta^K_{11} & & & \beta^K_{m_O 1} & & \\ & & \cdots & & & \\ & \beta^K_{1 m_D} & & & \beta^K_{m_O m_D} & \end{bmatrix}$$

with entries β_{ij}^k, for all $k \in < K >$, given by

$$\beta_{ij}^k = \begin{cases} 1, & \text{if } i \in \delta^-(j), \\ 0, & \text{otherwise.} \end{cases} \tag{18}$$

The remaining entries of D are all zeros. E is a generalized-upper-bound (GUB) constraint matrix of dimension $n \times nK$ of the form:

$$E = \begin{bmatrix} e^1_{11} & & & e^2_{11} & & & & e^K_{11} & & \\ & e^1_{12} & & & e^2_{12} & & & & e^K_{12} & \\ & & \ddots & & & \ddots & \cdots & & & \ddots \\ & & & e^1_{m_O m_D} & & & e^2_{m_O m_D} & & & e^K_{m_O m_D} \end{bmatrix}$$

with entries e_{ij}^k, for all $k \in < K >$, given by

$$e_{ij}^k = \begin{cases} 1, & \text{if } (i,j) \in \mathcal{E}, \\ 0, & \text{otherwise.} \end{cases} \tag{19}$$

The remaining entries of E are all zeros.

We also use

$$z = \begin{bmatrix} \pi^O \\ \pi^D \\ r \\ \psi \end{bmatrix}$$

to denote the vector of dual prices, where $\pi^O \in \Re^{m_O K}$ is the vector $(\pi_1^O \mid \pi_2^O \mid \ldots \mid \pi_K^O)^T$, $\pi^D \in \Re^{m_D K}$ is the vector $(\pi_1^D \mid \pi_2^D \mid \ldots \mid \pi_K^D)^T$, and $r \in \Re^{nK}$ is the vector $(r_1 \mid r_2 \mid \ldots \mid r_K)^T$.

Finally, let

$$\Phi = \begin{bmatrix} S \\ D \\ I \\ E \end{bmatrix},$$

$$\gamma = \begin{bmatrix} s \\ d \\ 0 \\ 0 \end{bmatrix},$$

$$\delta = \begin{bmatrix} s \\ d \\ u \\ U \end{bmatrix},$$

and $\phi^\ell = (\phi_t^\ell) = (\phi_{\ell t})$ denotes the ℓ-th column of Φ^T, the transpose of Φ. We use the lexicographic ordering of the variables, t, to index ϕ^ℓ, where the lexicographic order of variable x_{ij}^k is given by $t = (k-1)n + (i-1)m_O + j$. The rows of Φ are partitioned into four sets:

$I_O = \{l \mid l = m_D(k-1) + i, \forall i \in < m_O >, k \in < K >\}$, corresponding to rows of equality constraints over all origin nodes, (i.e., rows of equation (13)).

$I_D = \{l \mid l = m_O k + j, \forall j \in < m_D >, k \in < K >\}$, corresponding to rows of equality constraints over all destination nodes, (i.e., rows of equation (14)). Rows corresponding to both origin and destination nodes are denoted by $I_1 = I_O \cup I_D$.

$I_2 = \{l = (m_O + m_D)K + q, \forall q \in < n >\}$, corresponding to the simple bounds of equation (15).

$I_3 = \{l = (m_O + m_D + n)K + q, \forall q \in < n >\}$, corresponding to rows of the GUB constraints, (i.e., rows of equation (16)).

With this notation the pure multicommodity problem can be expressed as

$$\begin{aligned} &Minimize \quad F(x) && (20)\\ &Subject\ to: \\ &\gamma \leq \quad \Phi x \leq \quad \delta && (21) \end{aligned}$$

With slight modifications in the definitions of D, E, γ and δ, the same formulation expresses the generalized multicommodity problem. The modifications are the following:

1. Entries β_{ij}^k of the matrix D are given by

$$\beta_{ij}^k = \begin{cases} m_k(i,j), & \text{if } i \in \delta^-(j), \\ 0, & \text{otherwise.} \end{cases} \tag{22}$$

2. Entries e_{ij}^k of the matrix E are given by

$$e_{ij}^k = \begin{cases} e_k(i,j), & \text{if } (i,j) \in \mathcal{E}, \\ 0, & \text{otherwise.} \end{cases} \tag{23}$$

3. Define

$$\gamma = \begin{bmatrix} -\infty \\ d \\ 0 \\ 0 \end{bmatrix}.$$

We will be developing the algorithms starting from the compact matrix notation, but will be expressing them finally in the algebraic representation of equations (2)–(5) and (7) – (10). It is only in the later form that the fine-grain decomposition becomes apparent.

1.3 Outline of the Paper

In Section 2 we develop the algorithms. The general row-action framework is first summarized. It is then used to develop fine-grain decomposition algorithms for both pure and generalized problems and for both quadratic and entropy objective functions. Section 3 develops the data structures for the implementation of the algorithms on massively parallel computers of the SIMD (i.e., single instruction stream, multiple data stream) class. Section 4 presents results from numerical experiments conducted on a Connection Machine CM–2.

The experiments establish the efficiency of the algorithms for the solution of very large problems. They also provide some insight on the performance of the algorithms for different problem characteristics. Section 5 presents the conclusions of our study and discusses directions for further research.

2 The Fine-Grain Parallel Algorithms

We begin with a sketch of the main idea. A fine-grain decomposition of the multicommodity transportation problem is developed when properly applying a row-action iterative algorithm. Such an algorithm operates on one row of the constraint set at a time — hence its name. The iterative step consists of an adjustment of the dual price of the constraint followed by an adjustment of the primal variables. Throughout, complementarity is preserved and upon completion of the algorithm primal feasibility is achieved. The algorithm iterates using an almost-cyclic control sequence over all the constraints.

Obviously, if two constraints do not share any primal variables, the respective iterative steps can be executed independently. If multiple processors are available, they can also be executed in parallel. It is then desirable to use the almost-cyclic control mechanism to choose independent constraints. In the case of the multicommodity transportation problem, the algorithm first iterates on the origin nodes for all the commodities, then it iterates on the destination nodes for all the commodities, then it iterates on the simple bounds for all the commodities, and finally it iterates on the GUB constraints. One iteration of the algorithm requires $O(n + (m_O + m_D)K)$ operations. When iterating on the origin nodes it requires $m_O K \cdot O(1)$ operations, on the destination nodes it requires $m_D K \cdot O(1)$ operations, on the simple bounds it requires $n \cdot O(1)$ operations, and on the joint capacity constraints it requires $n \cdot O(1)$ operations. Hence, if the number of processors P scales linearly with $\max\{n,\ m_O K,\ m_D K\}$, the algorithm can be executed in constant number of operations per iteration.

The rest of this section makes these ideas precise.

2.1 The Row-Action Framework

We need some preliminary discussion. Let $F : \Lambda \subseteq \Re^n \rightarrow \Re$ and let $S \neq \phi$ be an open convex set such that $\bar{S} \subseteq \Lambda$. The set S is called the zone of F if F is strictly convex and continuous on $\bar{S}$ and continuously differentiable on S.

Let $D(x,y) = F(x) - F(y) - < \nabla F(y), x - y >$, and let $H(a,b)$ be the hyperplane $H(a,b) = \{x \in \Re^n \mid < a, x >= b\}$. The D-projection (or Bregman projection) of a point y

onto $H(a,b)$ is defined by

$$P_{H(a,b)} = \arg \min_{x \in H(a,b) \cap \bar{S}} D(x,y) \tag{24}$$

A function F — that belongs to the family of Bregman's functions as characterized by Censor and Lent [1981] — has the zone consistency property with respect to the hyperplane $H(a,b)$ if the D-projection of every $y \in S$ onto $H(a,b)$ is also in S. If a function is zone consistent with respect to $H(a,b)$ then it can be shown, Censor and Lent [1981, Lemma 3.1] that the D-projection of y onto $H(a,b)$ is the point x given by the unique solution of the nonlinear equations in x and β:

$$\nabla F(x) = \nabla F(y) + \beta \cdot a \tag{25}$$

$$< a, x > = b \tag{26}$$

The unique real number β is known as the Bregman parameter.

Both the quadratic and entropy functions [Q] and [E] are Bregman's functions, Censor and Lent [1981]. It is also easy to verify that — under Assumptions 1–3 — both functions have the strong zone consistency property with respect to the hyperplanes $H(\phi^\ell, \gamma_\ell)$ and $H(\phi^\ell, \delta_\ell)$ for all $\ell \in I_1 \cup I_2 \cup I_3$. Hence we can apply the following general iterative scheme:

Algorithm 2.1 General Row-Action Algorithm for Mixed Equality and Inequality Constraints

Step 0: (Initialization) $\nu \leftarrow 0$. Get z^0 and x^0 such that

$$\nabla F(x^0) = -\Phi^T z^0. \tag{27}$$

Step 1: (Iterative step over equality constraints). Choose a row index $\ell(\nu) \in I_1$.

$$\nabla F(x^{\nu+1/2}) = \nabla F(x^\nu) + \beta_\nu \phi^{\ell(\nu)} \tag{28}$$

$$z^{\nu+1/2} = z^\nu - \beta_\nu e^{\ell(\nu)}, \tag{29}$$

where β_ν is the Bregman parameter associated with the D-projection of x^ν on the hyperplane $H(\phi^{\ell(\nu)}, \gamma_{\ell(\nu)})$.

Step 2: (Iterative step over interval constraints). Choose a row index $\ell(\nu) \in I_2 \cup I_3$.

$$\nabla F(x^{\nu+1}) = \nabla F(x^{\nu+1/2}) + \beta_\nu \phi^{\ell(\nu)} \tag{30}$$

$$z^{\nu+1} = z^{\nu+1/2} - \beta_\nu e^{\ell(\nu)} \tag{31}$$

$$\beta_\nu = mid\{z^{\nu+1/2}_{\ell(\nu)}, \Gamma_\nu, \Delta_\nu\}. \tag{32}$$

Γ_ν and Δ_ν are the Bregman parameters associated with the D-projection of $x^{\nu+1/2}$ on the hyperplanes specified when the left and right inequalities, respectively, of the interval constraints hold with equality (i.e., $x^{\nu+1/2}$ is projected on the hyperplanes $H(\phi^{\ell(\nu)}, \gamma_{\ell(\nu)})$ and $H(\phi^{\ell(\nu)}, \delta_{\ell(\nu)})$ respectively). $\{\ell(\nu)\}$ is the control sequence of the algorithm, henceforth abbreviated as $\ell = \ell(\nu)$. $e^\ell \in \Re^{m_O+m_D+2n}$ is the $\ell - th$ standard basis vector having 1 in the $\ell - th$ coordinate and zeros elsewhere.

Step 3: Let $\nu \leftarrow \nu + 1$, and return to Step 1.

2.2 Algorithm for Quadratic Problems

We specialize now the general row-action scheme for quadratic programs with pure, multicommodity network constraints. The iterative step for the equality constraints is derived from Algorithm 2.1 (Step 1). It takes the form:

$$x_t^{\nu+1} = x_t^{\nu} + \frac{\beta_\nu}{w_t}\phi_t^{\ell},\ t = 1,2,\ldots Kn \tag{33}$$

$$z^{\nu+1} = z^{\nu} - \beta_\nu e^{\ell}, \tag{34}$$

where $\ell \in I_1$. The parameter β_ν is obtained by solving

$$y_t = x_t^{\nu} + \frac{\beta_\nu}{w_t}\phi_t^{\ell},\ t = 1,2,\ldots Kn \tag{35}$$

$$< \phi^{\ell}, y_t > = \gamma_\ell. \tag{36}$$

For any control index $\ell \in I_O$ the iterative step is obtained as follows: First — by the definition of I_O — ℓ can be expressed as $\ell = m_D(k-1) + i$ for some $i \in< m_O >$ and $k \in< K >$. Hence, ϕ^{ℓ} is the i-th row of the k-th block of matrix S with entries a_{ij}^k as given by (17). x_t^{ν} will be updated according to equation (33) only when $\phi_t^{\ell} \neq 0$. This occurs for values of t that correspond to the lexicographic ordering of variable $x_k(i,j)$ for $j \in \delta^+(i)$, since it is only for these values that $a_{ij}^k = 1$. Furthermore, the ℓ-th row of γ is $s_k(i)$ with dual variable $\pi_O^k(i)$. Hence, system (35)–(36) can be simplified to

$$y_k(i,j) = x_k^{\nu}(i,j) + \frac{\beta_\nu}{w_k(i,j)} \tag{37}$$

$$\sum_{j \in \delta^+(i)} y_k(i,j) = s_k(i), \tag{38}$$

for all $k \in< K >$. Solving this system for β_ν we obtain

$$\beta_\nu = \frac{1}{\sum_{j \in \delta^+(i)} \frac{1}{w_k(i,j)}} \left[s_k(i) - \sum_{j \in \delta^+(i)} x_k^{\nu}(i,j) \right]. \tag{39}$$

This expression for β_ν is substituted in (33)–(34) to complete the iterative step. Similarly, we obtain the iterative step for any control index $\ell \in I_D$.

The iterative step for the simple bound constraints (i.e., $\ell \in I_2$) can be obtained as a simplification of the interval constrained step. Is is identical to the step for the single commodity transportation problems, see Zenios and Censor [1990], and its derivation is not repeated here.

For any control index $\ell \in I_3$ the iterative step is taken over the interval constraints. It is obtained from Algorithm 2.1 (Step 2):

$$x_t^{\nu+1} = x_t^{\nu} + \frac{\beta^{\nu}}{w_t}\phi_t^{\ell},\ t = 1,2,\ldots kn \tag{40}$$

$$z^{\nu+1} = z^{\nu} - \beta_{\nu} e^{\ell}, \tag{41}$$

when $\beta_{\nu} = mid\{z_{\ell}^{\nu},\ \Gamma_{\nu},\ \Delta_{\nu}\}$. Γ_{ν} is obtained by solving

$$y_t = x_t^{\nu} + \frac{\Gamma_{\nu}}{w_t}\phi_t^{\ell},\ t = 1,2,\ldots Kn \tag{42}$$

$$< \phi^{\ell}, y > = \gamma_{\ell}, \tag{43}$$

and Δ_{ν} is obtained by solving

$$y_t = x_t^{\nu} + \frac{\Delta_{\nu}}{w_t}\phi_t^{\ell},\ t = 1,2,\ldots kn \tag{44}$$

$$< \phi^{\ell}, y > = \delta_{\ell}. \tag{45}$$

In order to solve for Γ_{ν} or Δ_{ν}, we need once more to examine the structure of ϕ^{ℓ} when $\ell \in I_3$ is given by $\ell = (m_O + m_D + n)K + q$ for some $q \in < n >$. ϕ^{ℓ} is the q-th row of the matrix E with entries e_{ij}^{k} as given by (19). x_t^{ν} will be updated according to (40) only when $\phi_t^{\ell} \neq 0$. This occurs for values of t that correspond to the lexicographic ordering of variables $x_k(i,j)$ that satisfy $q = m_O i + j$ for all $k \in < K >$. Furthermore, the ℓ-th row of γ is 0, the ℓ-th row of δ is $U(i,j)$, and the dual variable is $\psi(i,j)$. With these observations, systems (42)–(43) and (44)–(45) can be simplified to:

$$y_k(i,j) = x_k^{\nu}(i,j) + \frac{\Gamma_{\nu}}{w_k(i,j)} \tag{46}$$

$$\sum_{k=1}^{K} y_{k(i,j)} = 0, \tag{47}$$

and

$$y_k(i,j) = x_t^{\nu}(i,j) + \frac{\Delta_{\nu}}{w_k(i,j)} \tag{48}$$

$$\sum_{k=1}^{K} y_k(i,j) = U(i,j), \tag{49}$$

for all $k \in < K >$. Solving these systems, we obtain

$$\Gamma_\nu = -\frac{1}{\sum_{k=1}^{K} \frac{1}{w_k(i,j)}} \sum_{k=1}^{K} x_k^\nu(i,j) \tag{50}$$

and

$$\Delta_\nu = \frac{1}{\sum_{k=1}^{K} \frac{1}{w_k(i,j)}} \left[U(i,j) - \sum_{k=1}^{K} x_k^\nu(i,j) \right]. \tag{51}$$

These expressions for Γ_ν and Δ_ν are used to compute β_ν, which is then substituted in (40)–(41) to complete the iterative step.

We have now all the components required to complete the algorithm for pure multicommodoty transportation problems with a quadratic objective function:

Algorithm 2.2: Quadratic Optimization Algorithm for Pure Multicommodity Transportation Problems.

Step 0: (Initialization) $\nu \leftarrow 0$. $z^0 \leftarrow 0$,

$$x_k^0(i,j) = -\frac{c_k(i,j)}{w_k(i,j)}, \ \forall k \in < K >, \ (i,j) \in \mathcal{E}. \tag{52}$$

Step 1: (Solve the single commodity problems)

FOR $k = 1,2,3,\ldots K$:

Step 1.1: (Solve for origin nodes)

FOR $i = 1,2,3,\ldots m_O$:

Compute

$$\beta_\nu = \frac{1}{\sum_{j \in \delta^+(i)} \frac{1}{w_k(i,j)}} \left[s_k(i) - \sum_{j \in \delta^+(i)} x_k^\nu(i,j) \right] \tag{53}$$

Update

$$x_k^\nu(i,j) \leftarrow x_k^\nu(i,j) + \frac{\beta_\nu}{w_k(i,j)}, \ j \in \delta^+(i) \tag{54}$$

$$(\pi_k^O(i))^{\nu+1} = (\pi_k^0(i))^\nu - \beta_\nu \tag{55}$$

ENDFOR

Step 1.2: (Solve for destination nodes)

FOR $j = 1,2,3,\ldots m_D$:

Compute

$$\beta_\nu = \frac{1}{\sum_{i \in \delta^-(j)} \frac{1}{w_k(i,j)}} \left[d_k(j) - \sum_{i \in \delta^-(j)} x_k^\nu(i,j) \right] \tag{56}$$

Update

$$x_k^\nu(i,j) \leftarrow x_k^\nu(i,j) + \frac{\beta_\nu}{w_k(i,j)}, \ i \in \delta^-(j) \tag{57}$$

$$(\pi_k^D(i))^{\nu+1} = (\pi_k^D(j))^\nu - \beta_\nu \tag{58}$$

ENDFOR

Step 1.3: (Solve for the simple bounds)

FOR $(i,j) \in \mathcal{E}$:

Compute

$$\beta_\nu = mid\{r_k^\nu(i,j),\ w_k(i,j)\cdot(u_k(i,j) - x_k^\nu(i,j)),\ -w_k(i,j)\cdot x_k^\nu(i,j)\} \quad (59)$$

Update

$$x_k^\nu(i,j) \leftarrow x_k^\nu(i,j) + \frac{\beta_\nu}{w_k(i,j)}, \quad (60)$$

$$r_k^{\nu+1}(i,j) = r_k^\nu(i,j) - \beta_\nu \quad (61)$$

ENDFOR

ENDFOR

Step 2: (Solve for the joint capacity constraints)

FOR $(i,j) \in \mathcal{E}$:

Compute

$$\Gamma_\nu = -\frac{1}{\sum_{k=1}^K \frac{1}{w_k(i,j)}} \sum_{k=1}^K x_k^\nu(i,j) \quad (62)$$

$$\Delta_\nu = \frac{1}{\sum_{k=1}^K \frac{1}{w_k(i,j)}} \left[U(i,j) - \sum_{k=1}^K x_k^\nu(i,j)\right] \quad (63)$$

$$\beta_\nu = mid\{\psi^\nu(i,j), \Gamma_\nu, \Delta_\nu\} \quad (64)$$

Update

$$x_k^{\nu+1}(i,j) = x_k^\nu(i,j) + \frac{\beta_\nu}{w_k(i,j)},\ \forall k \in < K > \quad (65)$$

$$\psi^{\nu+1}(i,j) = \psi^\nu(i,j) - \beta_\nu,\ \forall k \in < K > . \quad (66)$$

ENDFOR

Step 3: Let $\nu \leftarrow \nu + 1$, and return to Step 1.

2.3 Algorithm for Entropy Problems

The development of the algorithm for entropy optimization problems proceeds along similar lines as the quadratic programming algorithm of the previous section. The basic iterative step for the equality constraints — see Step 1 of Algorithm 2.1 — is of the form:

$$x_t^{\nu+1} = x_t^{\nu} exp(\beta_\nu \phi_t^\ell) \tag{67}$$

$$z^{\nu+1} = z^{\nu} - \beta_\nu e^\ell, \tag{68}$$

where $\ell \in I_1$. The parameter β_ν is obtained by solving

$$y_t = x_t^{\nu} exp(\beta_\nu \phi_t^\ell) \tag{69}$$

$$< \phi^\ell, y > = \gamma_\ell. \tag{70}$$

We make the same observations as in section 2.2 about the structure of ϕ^ℓ and γ to obtain the system:

$$y_k(i,j) = x_k^{\nu}(i,j)\, exp(\beta_\nu) \tag{71}$$

$$\sum_{j \in \delta^+(i)} y_k(i,j) = s_k(i), \tag{72}$$

for all $k \in < K >$. Solving this system for $exp(\beta^\nu)$ we obtain:

$$exp(\beta^\nu) = \frac{s_k(i)}{\sum_{j \in \delta^+(i)} x_k^{\nu}(i,j)}. \tag{73}$$

This value of $exp(\beta^\nu)$ can be used in the primal updating step in (67). It appears, however, that in order to update the dual variables according to (68) we need β_ν. To avoid taking logarithms on (73), the algorithm will be working on the logarithm of the dual prices. If we let $\bar{z}_\ell = \ln z_\ell$ then the dual updating step is of the form:

$$\bar{z}_\ell^{\nu+1} = \frac{\bar{z}_\ell^{\nu}}{exp(\beta_\nu)}. \tag{74}$$

Hence, the iterative step can be completed using only algebraic operations.

Similarly, for the interval constraints, the algorithm takes the form:

$$x_t^{\nu+1} = x_t^{\nu}\, exp(\beta_\nu \phi_t^\ell) \tag{75}$$

$$z^{\nu+1} = z^{\nu} - \beta_\nu e^\ell, \tag{76}$$

where $\beta_\nu = mid\{z_\ell^\nu, \Gamma_\nu, \Delta_\nu\}$. Making the same observations as in section 2.2 about the structure of ϕ^ℓ, γ and δ, we arrive at the following system for Γ_ν and Δ_ν.

$$y_k(i,j) = x_k(i,j)^\nu exp(\Gamma_\nu) \tag{77}$$

$$\sum_{k=1}^{K} y_k(i,j) = 0, \tag{78}$$

and

$$y_k(i,j) = x_k^\nu(i,j)\, exp(\Delta_\nu) \tag{79}$$

$$\sum_{k=1}^{K} y_k(i,j) = U(i,j), \tag{80}$$

for all $k \in < K >$. It follows that

$$exp(\Gamma_\nu) = 0, \text{ and} \tag{81}$$

$$exp(\Delta_\nu) = \frac{U(i,j)}{\sum_{k+1}^{K} x_k^\nu(i,j)} \tag{82}$$

In order to avoid once more taking logarithms use the substitution $\bar{z}_\ell = \ln z_\ell$ and the fact that $\ln mid\,(\exp\alpha, \exp\beta, \exp\gamma) = mid(\alpha,\beta,\gamma)$. Then

$$exp(\beta_\nu) = mid\{\bar{z}_\ell^\nu, \exp(\Gamma_\nu), \exp(\Delta_\nu)\} \tag{83}$$

$$= min\{\bar{z}_\ell^\nu, \exp(\Delta_\nu)\}. \tag{84}$$

This expression for $\exp(\beta_\nu)$ is used in (75) for the primal updating step. The dual updating step is of the form

$$\bar{z}_\ell^{\nu+1} = \frac{\bar{z}_\ell^\nu}{exp(\beta_\nu)}. \tag{85}$$

Working out the algebra, we can now summarize the preceding discussion in the following:

Algorithm 2.3: Entropy Optimization Algorithm for Pure Multicommodity Transportation Problems.

Step 0: (Initialization) $\nu \leftarrow 0$. $\bar{z}^0 \leftarrow 1$,

$$x_k^0(i,j) = a_k(i,j),\ \forall k \in < K >,\ (i,j) \in \mathcal{E}. \tag{86}$$

Step 1: (Solve the single commodity problems)

FOR $k = 1,2,3,\ldots K$:

Step 1.1: (Solve for origin nodes)

FOR $i = 1,2,3,\ldots m_O$:

Compute

$$\beta_\nu = \frac{s_k(i)}{\sum_{j \in \delta^+(i)} x_k^\nu(i,j)} \tag{87}$$

Update

$$x_k^\nu(i,j) \leftarrow x_k^\nu(i,j)\beta_\nu,\ j \in \delta^+(i) \tag{88}$$

$$(\bar{\pi}_k^O(i))^{\nu+1} = \frac{(\bar{\pi}_k^0(i))^\nu}{\beta_\nu} \tag{89}$$

ENDFOR

Step 1.2: (Solve for destination nodes)

FOR $j = 1,2,3,\ldots m_D$:

Compute

$$\beta_\nu = \frac{d_k(j)}{\sum_{i \in \delta^-(j)} x_k^\nu(i,j)} \tag{90}$$

Update

$$x_k^\nu(i,j) \leftarrow x_k^\nu(i,j)\beta_\nu,\ i \in \delta^-(j) \tag{91}$$

$$(\bar{\pi}_k^D(i))^{\nu+1} = \frac{(\bar{\pi}_k^D(j))^\nu}{\beta_\nu} \tag{92}$$

ENDFOR

Step 1.3: (Solve for the simple bounds)

FOR $(i,j) \in \mathcal{E}$:

Compute

$$\beta_\nu = min\{\bar{r}_k^\nu(i,j),\ \frac{u_k(i,j)}{x_k^\nu(i,j)}\} \tag{93}$$

Update

$$x_k^\nu(i,j) \leftarrow x_k^\nu(i,j)\beta_\nu, \tag{94}$$

$$\bar{r}_k^{\nu+1}(i,j) = \frac{\bar{r}_k^\nu(i,j)}{\beta_\nu} \tag{95}$$

ENDFOR

ENDFOR

Step 2: (Solve for the joint capacity constraints)

FOR $(i,j) \in \mathcal{E}$:

Compute

$$\beta_\nu = \min\left\{\bar{\psi}^\nu(i,j), \frac{U(i,j)}{\sum_{k=1}^K x_k^\nu(i,j)}\right\} \tag{96}$$

Update

$$x_k^{\nu+1}(i,j) = x_k^\nu(i,j)\beta_\nu,\ \forall k \in < K > \tag{97}$$

$$\bar{\psi}^{\nu+1}(i,j) = \frac{\bar{\psi}^\nu(i,j)}{\beta_\nu},\ \forall k \in < K > . \tag{98}$$

ENDFOR

Step 3: Let $\nu \leftarrow \nu + 1$, and return to Step 1.

2.4 Extensions to Generalized Networks

The algorithms of sections 2.2 and 2.3 can be extended to the generalized problem [GMTR]. For the quadratic programming problem the development of the algorithm follows the discussion of section 2.2. The algorithm for the entropy problem does not have closed-form solutions for the Bregman parameters. It turns out, however, that an approximation to these parameters can be computed in closed form without destroying the asymptotic convergence of the algorithm.

2.4.1 Algorithm for Quadratic Problems

In order to develop the algorithm, based on the developments of section 2.2 we need to make three observations. First, the iterative steps when the control sequence is taken from I_O is a step over an interval constraint. (Recall that $\gamma_\ell = -\infty$ and $\delta_\ell = s_\ell$.) Second, for control sequence $\ell \in I_D$ the coefficients of ϕ^ℓ are given by equation (22). Hence, the generalized network multipliers $\{m_k(i,j)\}$ enter the calculation. Third, the entries of ϕ^ℓ for $\ell \in I_3$ are given by equation (23). Hence, the coefficients $\{e_k(i,j)\}$ enter the calculation. Using now the basic Algorithm 2.1, much in the same way as we did in section 2.2, we obtain:

Algorithm 2.4: Quadratic Optimization Algorithm for Generalized Multicommodity Transportation Problems.

Step 0: (Initialization) $\nu \leftarrow 0$. $z^0 \leftarrow 0$,

$$x_k^0(i,j) = -\frac{c_k(i,j)}{w_k(i,j)}, \ \forall k \in < K >, \ (i,j) \in \mathcal{E}. \tag{99}$$

Step 1: (Solve the single commodity problems)

FOR $k = 1,2,3,\ldots K$:

Step 1.1: (Solve for origin nodes)

FOR $i = 1,2,3,\ldots m_O$:

Compute

$$\Delta_\nu = \frac{1}{\sum_{j\in\delta^+(i)} \frac{1}{w_k(i,j)}} \left[s_k(i) - \sum_{j\in\delta^+(i)} x_k^\nu(i,j) \right] \tag{100}$$

$$\beta_\nu = \min\{(\pi_k^O(i,j))^\nu, \ \Delta_\nu\} \tag{101}$$

Update

$$x_k^\nu(i,j) \leftarrow x_k^\nu(i,j) + \frac{\beta_\nu}{w_k(i,j)}, \ j \in \delta^+(i) \tag{102}$$

$$(\pi_k^O(i))^{\nu+1} = (\pi_k^0(i))^\nu - \beta_\nu \tag{103}$$

ENDFOR

Step 1.2: (Solve for destination nodes)

FOR $j = 1,2,3,\ldots m_D$:

Compute

$$\beta_\nu = \frac{1}{\sum_{i\in\delta^-(j)} \frac{m_k^2(i,j)}{w_k(i,j)}} \left[d_k(j) - \sum_{i\in\delta^-(j)} m_k(i,j) \cdot x_k^\nu(i,j) \right] \tag{104}$$

Update

$$x_k^\nu(i,j) \leftarrow x_k^\nu(i,j) + \frac{m_k(i,j)}{w_k(i,j)} \cdot \beta_\nu, \ i \in \delta^-(j) \tag{105}$$

$$(\pi_k^D(i))^{\nu+1} = (\pi_k^D(j))^\nu - \beta_\nu \tag{106}$$

ENDFOR

Step 1.3: (Solve for the simple bounds)

FOR $(i,j) \in \mathcal{E}$:

Compute

$$\beta_\nu = mid\{r_k^\nu(i,j),\ w_k(i,j)\cdot(u_k(i,j) - x_k^\nu(i,j)),\ -w_k(i,j)\cdot x_k^\nu(i,j)\} \quad (107)$$

Update

$$x_k^\nu(i,j) \leftarrow x_k^\nu(i,j) + \frac{\beta_\nu}{w_k(i,j)}, \quad (108)$$

$$r_k^{\nu+1}(i,j) = r_k^\nu(i,j) - \beta_\nu \quad (109)$$

ENDFOR

ENDFOR

Step 2: (Solve for the joint capacity constraints)

FOR $(i,j) \in \mathcal{E}$:

Compute

$$\Gamma_\nu = -\frac{1}{\sum_{k=1}^K \frac{e_k^2(i,j)}{w_k(i,j)}} \sum_{k=1}^K e_k(i,j)\cdot x_k^\nu(i,j) \quad (110)$$

$$\Delta_\nu = \frac{1}{\sum_{k=1}^K \frac{e_k^2(i,j)}{w_k(i,j)}} \left[U(i,j) - \sum_{k=1}^K e_k(i,j)\cdot x_k^\nu(i,j)\right] \quad (111)$$

$$\beta_\nu = mid\{\psi^\nu(i,j), \Gamma_\nu, \Delta_\nu\} \quad (112)$$

Update

$$x_k^{\nu+1}(i,j) = x_k^\nu(i,j) + \frac{e_k(i,j)}{w_k(i,j)}\cdot\beta_\nu,\ \forall k \in < K > \quad (113)$$

$$\psi^{\nu+1}(i,j) = \psi^\nu(i,j) - \beta_\nu,\ \forall k \in < K > . \quad (114)$$

ENDFOR

Step 3: Let $\nu \leftarrow \nu + 1$, and return to Step 1.

2.4.2 Algorithm for Entropy Problems

Finally, we develop the entropy optimization algorithm for the generalized problems. The basic iterative step for control sequence parameter $\ell \in I_O$ is of the form:

$$x_t^{\nu+1} = x_t^{\nu} exp(\beta_\nu \phi_t^\ell) \quad (115)$$

$$z^{\nu+1} = z^{\nu} - \beta_\nu e^\ell, \quad (116)$$

where $\beta_\nu = \min\{z_\ell^\nu, \Delta_\nu\}$ and Δ_ν is the solution of:

$$y_t = x_t^{\nu} exp(\Delta_\nu \phi_t^\ell) \quad (117)$$

$$< \phi^\ell, y > = \delta_\ell. \quad (118)$$

We make the same observations as in section 2.3 about ϕ^ℓ and δ_ℓ to obtain the system:

$$y_k(i,j) = x_k^\nu(i,j) \exp(\Delta_\nu) \quad (119)$$

$$\sum_{j \in \delta^+(i)} y_k(i,j) = s_k(i), \quad (120)$$

for all $k \in < K >$. Solving for $\exp(\Delta^\nu)$ we obtain:

$$\exp(\Delta^\nu) = \frac{s_k(i)}{\sum_{j \in \delta^+(i)} x_k^\nu(i,j)}. \quad (121)$$

We make once more the transformation of dual variables $\bar{z}_\ell = \ln z_\ell$, and using the fact that $\ln \min(\exp(\alpha), \exp(\beta)) = \min(\alpha, \beta)$ we can use

$$\exp(\beta_\nu) = \min(\bar{z}_\ell, \exp(\Delta_\nu)) \quad (122)$$

This value of $exp(\beta^\nu)$ is used directly in (116) for the primal updating step. The dual updating step becomes

$$\bar{z}_\ell^{\nu+1} = \frac{\bar{z}_\ell^\nu}{exp(\beta_\nu)}. \quad (123)$$

The iterative step for $\ell \in I_D$ is obtained along similar lines. Recall that — by the definition of I_D — ℓ can be expressed as $\ell = m_O k + j$ for some $j \in < m_D >$ and $k \in < K >$. Hence, ϕ^ℓ is the j-th row of the k-th block of matrix D, with entries β_{ij}^k as given by (22). The Bregman parameter β_ν is obtained from:

$$y_k(i,j) = x_k^\nu(i,j) \exp(\beta_\nu m_k(i,j)) \quad (124)$$

$$\sum_{i \in \delta^-(j)} m_k(i,j) y_k(i,j) = d_k(j). \quad (125)$$

Let $\omega_\nu = \exp(\beta_\nu)$ and rewrite this system as a nonlinear equation in ω_ν:

$$\varphi(\omega_\nu) \doteq \sum_{i \in \delta^-(j)} m_k(i,j) \cdot x_k^\nu(i,j) \cdot \omega_\nu^{m_k(i,j)} - d_k(j) = 0. \tag{126}$$

The existence of a solution to this equation follows from Lemma 1 of Censor et al. [1989]. However, its solution requires the use of an iterative procedure — such as the massively parallel linesearch algorithms developed by Zenios and Nielsen [1990]. It is possible, however, to obtain an approximate solution in closed form by taking one secant step: Consider the line through $(0, -d_k(j))$ and $(1, \varphi(1))$ instead of the graph of $\varphi(\omega_\nu)$. This line intersects the ω_ν - axis at

$$\hat{\omega}_\nu = \frac{d_k(j)}{\sum_{i \in \delta^-(j)} m_k(i,j) \cdot x_k^\nu(i,j)} \tag{127}$$

and we use this value of $\hat{\omega}_\nu$ as an approximate solution to (126). It can be shown — see Censor et al. [1989] — that using this approximation preserves asymptotic convergence of the general Algorithm 2.1. This value of $\hat{\omega}_\nu = \exp(\beta_\nu)$ is used in the primal updating step in (115). For the dual updating step we work once more in the space of $\bar{z}_\ell = \ln z_\ell$.

The iterative step for $\ell \in I_3$ is

$$x_t^{\nu+1} = x_t^\nu exp(\beta_\nu \phi_t^\ell) \tag{128}$$

$$z^{\nu+1} = z^\nu - \beta_\nu e^\ell \tag{129}$$

where $\beta_\nu = mid\{z_\ell^\nu, \Gamma_\nu, \Delta_\nu\}$. By the definition of I_3, ℓ can be expressed in the form $\ell = (m_O + m_D + n)K + q$ for some $q \in < n >$, and ϕ^ℓ is the q-th row of matrix E. Its entries are given by equation (23). Hence, the parameters Γ_ν and Δ_ν can be obtained by solving

$$y_k(i,j) = x_k^\nu(i,j) \exp(\Gamma_\nu e_k(i,j)) \tag{130}$$

$$\sum_{k=1}^{K} e_k(i,j) \cdot y_k(i,j) = 0, \tag{131}$$

and

$$y_k(i,j) = x_k^\nu(i,j) \exp(\Delta_\nu e_k(i,j)) \tag{132}$$

$$\sum_{k=1}^{K} e_k(i,j) \cdot y_k(i,j) = U(i,j). \tag{133}$$

The solution to the first system is $\exp(\Gamma_\nu) = 0$. For the second system we take a single secant step to get an approximate solution

$$\exp(\Delta_\nu) = \frac{U(i,j)}{\sum_{k=1}^{K} e_k(i,j) \cdot x_k^\nu(i,j)}. \tag{134}$$

We work once more in the transformed dual variable space $\bar{z}_\ell = \ln z_\ell$ and observe that

$$\begin{aligned} \exp(\beta_\nu) &= mid\{\bar{z}_\ell, \exp\Gamma_\nu, \exp\Delta_\nu\} \\ &= \min\{\bar{z}_\ell, \exp\Delta_\nu\}. \end{aligned}$$

Hence, we have completed all the components required to write down the entropy optimization algorithm:

Algorithm 2.5: Entropy Optimization Algorithm for Generalized Multicommodity Transportation Problems.

Step 0: (Initialization) $\nu \leftarrow 0$. $\bar{z}^0 \leftarrow 1$,

$$x_k^0(i,j) = a_k(i,j),\ \forall k \in < K >,\ (i,j) \in \mathcal{E}. \tag{135}$$

Step 1: (Solve the single commodity problems)

FOR $k = 1,2,3,\ldots K$:

Step 1.1: (Solve for origin nodes)

FOR $i = 1,2,3,\ldots m_O$:

Compute

$$\Delta_\nu = \frac{s_k(i)}{\sum_{j\in\delta^+(i)} x_k^\nu(i,j)} \tag{136}$$

$$\beta_\nu = \min\{\bar{\pi}_k^O(i), \Delta_\nu\} \tag{137}$$

Update

$$x_k^\nu(i,j) \leftarrow x_k^\nu(i,j)\beta_\nu,\ j \in \delta^+(i) \tag{138}$$

$$(\bar{\pi}_k^O(i))^{\nu+1} = \frac{(\bar{\pi}_k^O(i))^\nu}{\beta_\nu} \tag{139}$$

ENDFOR

Step 1.2: (Solve for destination nodes)

FOR $j = 1,2,3,\ldots m_D$:

Compute

$$\beta_\nu = \frac{d_k(j)}{\sum_{i\in\delta^-(j)} m_k(i,j)\cdot x_k^\nu(i,j)} \tag{140}$$

Update

$$x_k^\nu(i,j) \leftarrow x_k^\nu(i,j)\beta_\nu,\ i \in \delta^-(j) \tag{141}$$

$$(\bar{\pi}_k^D(i))^{\nu+1} = \frac{(\bar{\pi}_k^D(j))^\nu}{\beta_\nu} \tag{142}$$

ENDFOR

Step 1.3: (Solve for the simple bounds)

FOR $(i,j) \in \mathcal{E}$:

Compute

$$\beta_\nu = min\{\bar{r}_k^\nu(i,j),\ \frac{u_k(i,j)}{x_k^\nu(i,j)}\} \tag{143}$$

Update

$$x_k^\nu(i,j) \leftarrow x_k^\nu(i,j)\beta_\nu, \tag{144}$$

$$\bar{r}_k^{\nu+1}(i,j) = \frac{\bar{r}_k^\nu(i,j)}{\beta_\nu} \tag{145}$$

ENDFOR

ENDFOR

Step 2: (Solve for the joint capacity constraints)

FOR $(i,j) \in \mathcal{E}$:

Compute

$$\beta_\nu = \min\left\{\bar{\psi}^\nu(i,j), \frac{U(i,j)}{\sum_{k=1}^K e_k(i,j)\cdot x_k^\nu(i,j)}\right\} \tag{146}$$

Update

$$x_k^{\nu+1}(i,j) = x_k^\nu(i,j)\beta_\nu,\ \forall k \in < K > \tag{147}$$

$$\bar{\psi}^{\nu+1}(i,j) = \frac{\bar{\psi}^\nu(i,j)}{\beta_\nu},\ \forall k \in < K > . \tag{148}$$

ENDFOR

Step 3: Let $\nu \leftarrow \nu + 1$, and return to Step 1.

2.5 Discussion

Several issues deserve further analysis with respect to the developed algorithms. We discuss here some of these issues, with the objective to provide additional insight and provoke further studies.

2.5.1 Relaxation Parameters

Relaxation parameters $\{\lambda^\nu\}$ can be built into all the algorithms. Actually a different relaxation parameter can be used for different constraints, and they can also change value as the algorithm proceeds, provided they are within some limits imposed to guarantee convergence. For the quadratic algorithms, for example, $0 < \epsilon \leq \lambda^\nu < 2$ and for the entropy algorithms $0 < \epsilon \leq \lambda^\nu < 1$. The use of relaxation parameters could accelerate significantly convergence of the algorithm to an approximate solution. Since the algorithms are usually terminated when a sufficiently good solution is obtained, the use of relaxation parameters is of great practical significance. This is a topic worth further study.

2.5.2 Choice of Control Sequence

The algorithms developed here use the following control sequence: (1) origin node constraints, (2) destination node constraints, (3) simple bounds, (4) GUB constraints. Of course, any other almost cyclic control sequence will do. It is unclear which control sequence would accelerate the convergence of the algorithm towards an approximate solution. In addition to empirical studies, it is possible to undertake a more fundamental analysis: The algorithms project the current iterate x^ν on successive hyperplanes that are specified by the choice of $\phi^{\ell(\nu)}$. If $\phi^{\ell(\nu)}$ and $\phi^{\ell(\nu+1)}$ are almost parallel the algorithm will take very small steps. Looking at the cosine of the angle between successive hyperplanes

$$\frac{< \phi^{\ell(\nu)}, \phi^{\ell(\nu+1)} >}{\| \phi^{\ell(\nu)} \| \cdot \| \phi^{\ell(\nu+1)} \|} \tag{149}$$

we can choose hyperplanes that are (almost) orthogonal. On the other hand, if some hyperplanes $\{\phi^\ell\}$ are almost parallel, they could be replaced by a surrogate hyperplane. Given the rich structure of the constraint matrix Φ, it is worthwhile to investigate specialized acceleration schemes, starting from the general discussion of Bjorg and Elfving [1979] or Bramley and Sameh [1990].

2.5.3 Potentially Difficult Problems

It is possible to gain insight about the performance of the algorithm on a candidate class of test problems by examining the problem data and the structure of the algorithms. Refer, for example, to the quadratic programming Algorithm 2.4. In the primal and dual step calculation (Step 1.1), we see the term $\frac{1}{\sum_{j\in\delta^+(i)}\frac{1}{w_k(i,j)}}$. If the coefficients $w_k(i,j)$ are very small, so will be the whole term and the algorithm will be taking very small steps. Similar performance will be observed for very large and dense problems. In Step 1.2, we have the expression $\frac{1}{\sum_{i\in\delta^-(j)}\frac{m_k^2(i,j)}{w_k(i,j)}}$. If the problem multipliers are very large, the algorithm will be taking very small steps. On the other hand, if the multipliers are very small the algorithm will be taking very large steps in satisfying the destination node constraints. Such steps will, most likely, produce large errors in the origin node constraints and joint capacity constraints. Multipliers close to 1.0 would be the best for our algorithms. Models that use the multipliers to indicate gains/losses fit in this requirement For example, network models for financial planning and for water distribution systems have multipliers close to 1.0. Similar observations can be made on the structure of the coefficients $e_k(i,j)$.

2.5.4 Asymptotic Convergence

Since the algorithms we developed are specializations of the general row-action framework, their asymptotic convergence can be derived from known results. For pure problems and generalized quadratic problems, we can obtain convergence from the results of Censor and Lent [1981] and Elfving's [1989] extension to mixed equality and inequality constraints. For the generalized entropy problem, we can obtain convergence from the results of Censor et al. [1990]. For the relaxed versions of the algorithm, one needs to extend the results of Pierro and Iusem [1986] and Censor et al. [1989] to the mixed equality and internal constrained problem. Such extension is easy, see, for example, Elfving [1989].

3 Massively Parallel Implementations

The motivation for the design of the algorithms has been the desire to exploit massively parallel computing for the solution of very large problems. Of particular interest is the concept of data-level parallelism, whereby the problem is decomposed into fine-grain identical operations executed on multiple data. If a large number of processors is available, then each one could execute these operations on its local data elements.

When there is interaction among the problem data, it would be necessary to communicate among the corresponding processors. Such communication can be combined with computations, as for example when P processors with local data α_i, $i = 1, 2, 3, \ldots, P$ coordinate to compute the partial sums $\alpha_j = \sum_{\ell=1}^{j} a_i$. Or the communication step could be void of any computing, as in the case of permuting the data among processors according to some index list $\alpha_i \leftarrow \alpha_{list(i)}$.

The algorithms we developed decompose naturally for this form of parallelism. In this section we discuss implementations of Algorithm 2.2 on a massively parallel Connection Machine CM–2. However, the data structures developed here could be used to implement any one of the algorithms in this paper.

3.1 The Connection Machine CM–2

In this section we introduce the characteristics of the Connection Machine (model CM–2) that are relevant to the parallel implementations discussed in the sequel. Parts of this description were included in earlier reports and are presented here to make the paper self contained. Further details on the architecture of the CM can be found in Hillis [1985].

The Connection Machine is a fine grain SIMD — Single Instruction stream, Multiple Data stream — system. Its basic hardware component is an integrated circuit with sixteen processing elements (PEs) and a *router* that handles general communication. A fully configured CM has 4,096 chips for a total of 65,536 PEs. The 4,096 chips are interconnected as a 12-dimensional hypercube. Each processor is equipped with local memory of 8Kbytes, and for each cluster of 32 PEs a floating point accelerator handles floating point arithmetic.

Operations by the PEs are under the control of a *microcontroller* that broadcasts instructions from a front-end computer (FE) simultaneously to all the elements for execution. A flag register at every PE allows for no-operations; i.e., an instruction received from the microcontroller is executed if the flag is set, and ignored otherwise.

Parallel computations on the CM are in the form of a single operation executed on multiple copies of the problem data. All processors execute identical operations, each one operating on data stored in its local memory, accessing data residing in the memory of other PEs, or receiving data from the front end. This mode of computation is termed *data level parallelism* in contradistinction to *control level parallelism* whereby multiple processors execute their own control sequence, operating either on local or shared data.

To achieve high performance with data level parallelism one needs a large number of processors that could operate on multiple copies of the data concurrently. While the full configuration of the CM has 65,536 PEs this number is not large enough for several appli-

cations. The CM provides the mechanism of *virtual processors* (VPs) that allows one PE to operate in a serial fashion on multiple copies of data. VPs are specified by slicing the local memory of each PE into equal segments and allowing the physical processor to loop over all slices. The number of segments is called the *VP ratio* (i.e., ratio of virtual to physical PEs). Looping by the PE over all the memory slices is executed, in the worst case, in linear time. The set of virtual processors associated with each element of a data set is called a *VP set.* VP sets are under the control of the software and are mapped onto the underlying CM hardware in a way that is transparent to the user.

The CM supports two addressing mechanisms for communication. The *send* address is used for general purpose communications via the routers. The NEWS address describes the position of a VP in an n-dimensional grid that optimizes communication performance.

The *send* address indicates the location of the PE (hypercube address) that supports a specific VP and the relative address of the VP in the VP set that is currently active. NEWS address is an n-tuple of coordinates which specifies the relative position of a VP in an n-dimensional Cartesian-grid geometry. A *geometry* (defined by the software) is an abstract description of such an n-dimensional grid. Once a geometry is associated with the currently active VP set a relative addressing mechanism is established among the processors in the VP set. Each processor has a relative position in the n-dimensional geometry and NEWS allows the communication across the North, East, West and South neighbors of each processor, and enables the execution of operations along the axes of the geometry. Such operations are efficient since the n-dimensional geometry can be mapped onto the underlying hypercube in such a way that adjacent VPs are mapped onto vertices of the hypercube connected with a direct link. This mapping of an n-dimensional mesh on a hypercube is achieved through a Gray coding.

3.1.1 Elements of the Parallel Instruction Set Paris

Paris is the lowest level protocol by which the actions of the data processors of the CM are controlled by the front end. Interfaces with languages like C, Fortran or Lisp allow users to develop a program in a high-level language and then use Paris instructions to control the execution of parallel operations. Paris supports operations on signed, unsigned and floating-point numbers, message passing operations both along *send* and NEWS addresses and mechanisms for transferring data between the host and the data processors.

Before invoking Paris instructions from a program the user has to specify the VP set, create a geometry, and associate the VP set with the geometry. Thus a communications mechanism is established (along both *send* and NEWS addresses). Paris instructions —

parallel primitives — can then be invoked to execute operations along some axis of the geometry (using NEWS addresses), operate on an individual processor using *send* addresses, or to translate NEWS to *send* addresses for general interprocessor communication or communication with the front end. Parallel primitives that are relevant to our implementation are the *scans* and *spreads* of Blelloch [1990].

Scan is also known in the literature as parallel prefix. The $\otimes$-scan primitive, for an associative, binary operator $\otimes$, takes a sequence $\{x_0, x_1, \ldots, x_n\}$ and produces another sequence $\{y_0, y_1, \ldots, y_n\}$ such that $y_i = x_0 \otimes x_1 \otimes \cdots \otimes x_i$. On the Connection Machine, for example, *add-scan* takes as an argument a parallel variable (i.e., a variable with its $i-th$ element residing in a memory field of the $i-th$ VP) and returns at VP i the value of the parallel variable summed over $j = 0, \ldots, i$. User options allow the scan to apply only to preceding processors (e.g., sum over $j = 0, \ldots, i-1$) or to perform the scan in reverse. The $\otimes$-spread primitive, for an associative, binary operator $\otimes$, takes a sequence $\{x_0, x_1, \ldots, x_n\}$ and produces another sequence $\{y_0, y_1, \ldots, y_n\}$ such that $y_i = x_0 \otimes x_1 \otimes \cdots \otimes x_n$. For example, *add-spread* takes as an argument a parallel variable residing at the memory of n active data processors and returns at VP i the value of the parallel variable summed over $j = 0, \ldots, n$. An add-spread is equivalent to an add-scan followed by a reverse-copy-scan but is more efficient.

Another variation of the scan primitives allows their operation within *segments* of a parallel variable or VP. These primitives are denoted as *segmented-$\otimes$-scan.* They take as arguments a parallel variable and a set of segment bits which specify a partitioning of the VP set into contiguous segments. Segment bits have a 1 at the starting location of a new segment and a 0 elsewhere. A *segmented-$\otimes$-scan* operation restarts at the beginning of every segment. When processors are configured as a NEWS grid, scans within rows or columns are special cases of segmented scans called *grid-scans.*

3.2 Dense Implementation

In all implementations we assume that the individual commodities are unbounded (i.e., $u_k(i,j) = +\infty$, for all arcs and all commodities), and are only restricted through the GUB constraints. This is a reasonable practical assumption, and our implementations can be easily extended to remove this restriction.

In the dense implementation of the algorithm, it is assumed that the graph $\mathcal{G}$ is dense. The CM–2 is configured as a two-dimensional NEWS grid of dimensions $\lceil m_O \rceil_2 \times \lceil m_D \rceil_2$. This grid is then used to solve a sequence of single-commodity transportation problems, implemented as explained in Zenios and Censor [1990]. The memory of VP with NEWS

coordinates (i,j) stores the data for arc $(i,j) \in \mathcal{E}$. It is partitioned into the following data fields:

1. Supply and demand, s=$s_k(i)$ and d=$d_k(j)$.
2. Current iterate, x =$x_k(i,j)$.
3. Dual price PSI=$\psi(i,j)$.
4. Sum of the flows over all commodities ex=$\sum_{k=1}^{K} x_k(i,j)$.
5. Joint capacity constraint U=$U(i,j)$.
6. Three fields IW, JW and KW that hold the constants $\frac{1}{\sum_{i\in\delta^-(j)} \frac{1}{w_k(i,j)}}$, $\frac{1}{\sum_{j\in\delta^+(i)} \frac{1}{w_k(i,j)}}$ and $\frac{1}{\sum_{k=1}^{K} \frac{1}{w_k(i,j)}}$ respectively, and a field W that holds the quadratic coefficient.
7. Scaling factor BETA.
8. Scratch fields to hold intermediate results.

In the memory of the FE we define a vector of K structures of the form

```
a-commodity {
fe-w[m_O][m_D]
fe-x[m_O][m_D]
fe-s[m_O]
fe-d[m_D] } commodity-t;
commodity-t com[K] ;
```

Figure 1 illustrates the memory configuration of both the CM and the FE. With this layout of memory, the algorithm is executed as follows:

Step 0: Initialize according to Algorithm 2.2 (Step 0), and set BETA=0.

Step 1: Initialize ex to zero. Move one commodity at a time from the FE Update x and PSI according to (65) and (66), and solve by executing iteratively Steps 1.1–1.3 of Algorithm 2.2. Accumulate the optimal solution into field ex, and move the optimal solution from the CM to the FE.

Step 2: Compute the scaling factor (Step 2 of Algorithm 2.2, equation (64)), store in BETA and return to Step 1.

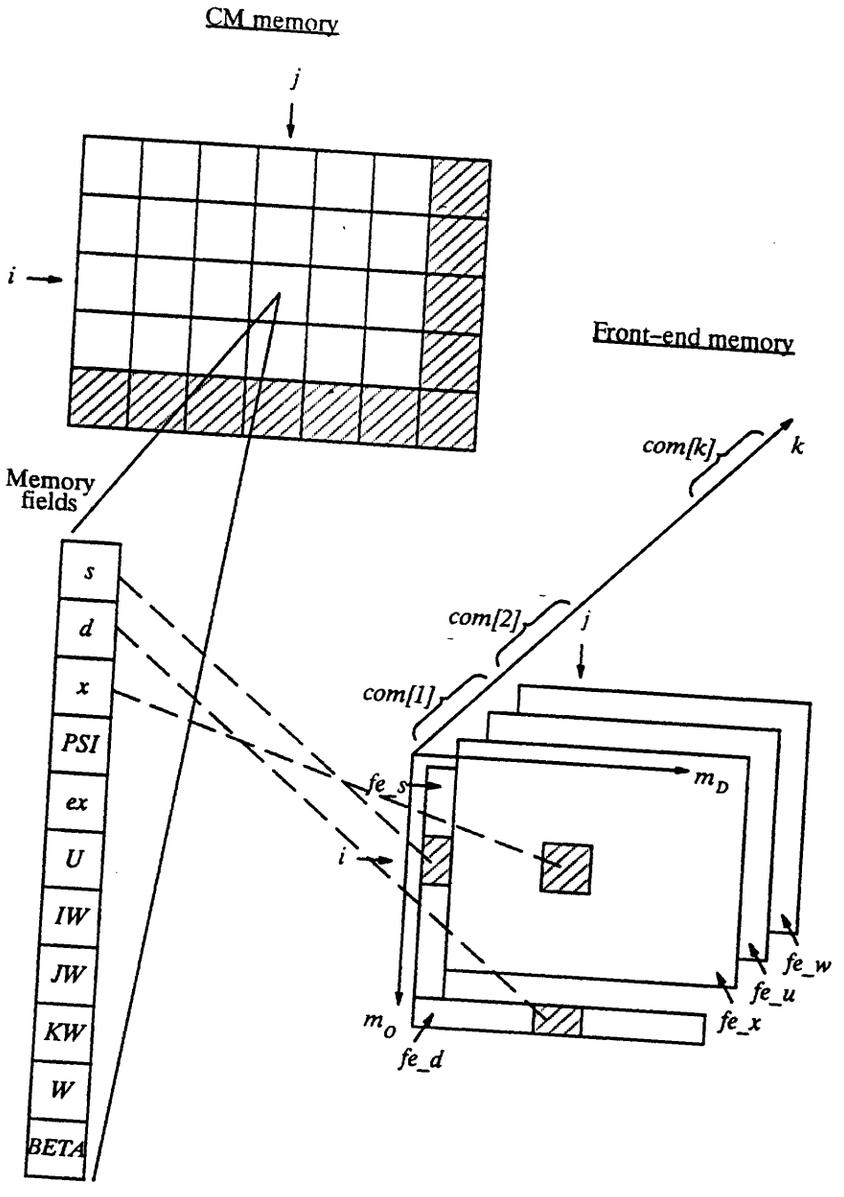


Figure 1: Memory configuration of the FE and the CM for the dense implementation.

3.3 Sparse Implementation

Solving sparse network optimization problems on the CM is particularly challenging. The arbitrary network topology has to be mapped to the virtual processors in a way that is efficient both for computations and communications. It appears that the data structures introduced in Zenios and Lasken [1989] are at present the best known method to represent sparse network problems. A comparison of alternative parallel implementations is reported in Nielsen and Zenios [1990b], and these data structures have been used by Eckstein [1990] for the implementation of his alternating directions method of multipliers with very encouraging results, and in the network optimization solver of Zenios and Nielsen [1990]. The representation adopted in these studies uses a 1-dimensional geometry of size $2n + (m_O + m_D)$. It assigns two VPs for each arc (i,j), one at the tail node i, and one at the head node j and one VP for each node. VPs that correspond to the same node are grouped together into a contiguous segment. In this way segmented-scan operations can be used for computing and for communicating data among processors incident to a node. The general communication of prices among nodes is a one-to-one send operation between the VPs at the head and tail of each arc.

In order to implement a sparse, multicommodity network solver, we use the single-commodity nonlinear network optimizer of Zenios and Nielsen [1990]. This solver is designed to handle sparse, transhipment problems. A further specialization for the bipartite graphs is also possible, but is not implemented here. Interestingly, the single commodity solver can be easily extended to solve multiple independent commodities in parallel: The CM is configured as a two-dimensional NEWS geometry, of dimensions $\lceil K \rceil_2 \times \lceil 2n+(m_O+m_D) \rceil_2$. Each row of the 0-axis is used to represent a single network problem as outlined above. Since the network problem has identical topology for all the commodities, the mapping of arcs into VPs and the partitioning of VPs into segments will be identical for each row of the NEWS axis.

The control of the algorithm is identical for each row of the NEWS grid (i.e., for each network problem). Row k of the 0-axis will store the data of the network problem for the k-th commodity. This configuration is illustrated in Figure 2. The algorithm iterates along the 1-axis until some convergence criteria is satisfied for all the rows. Once the single commodity networks are solved by iterations along the 1-axis, the algorithm executes Step 2 using scan operations along the 0-axis. (Note that, since the flows of each commodity satisfy $x_k(i,j) \geq 0$ we only need to compute the projection for the upper bound of the GUB constraints). This step is implemented by the following code segment:

```
CM-spread-with-f-add-1L(scr1, x, 0, S, E);
```

```
CM-f-sub-mult-1L(scr2, U, scr1, KW, S, E);
CM-f-min-2-1L(scr2, PSI, S, E);
CM-f-subtract-2-1L(PSI, scr2, S, E);
CM-f-divide-2-1L(scr2, W, S, E);
CM-f-add-2-1L(x, scr2, S, E);
```

4 Experimental Design and Performance Evaluation

The quadratic optimization Algorithm 2.2 was implemented on the Connection Machine CM–2 using C/Paris, as explained in the previous section. As pointed out in Section 2, the steps of Algorithm 2.2 can be executed in any almost-cyclic fashion. Our implementation carries out iteratively Steps 1.1–1.3 until some termination criterion is satisfied for the equality constraints (MINOR iterations). Once this tolerance is achieved, it executes Step 2 (MAJOR iteration) and resumes minor iterations. The algorithm terminates when both of the following termination criteria are satisfied:

1. Relative error on GUB constraints for major iterations:

$$100 \times \max_{(i,j)\in\mathcal{E}} \left\{ 0, \frac{\sum_{x=1}^{K} x_k(i,j) - U(i,j)}{U(i,j)} \right\} \leq \epsilon_1 \tag{150}$$

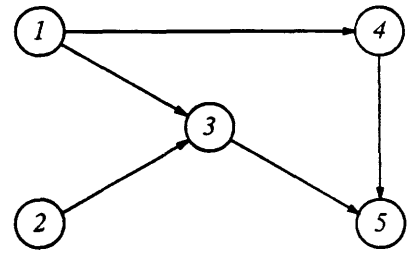
   In all experiments we set $\epsilon_1 = 0.1\%$.

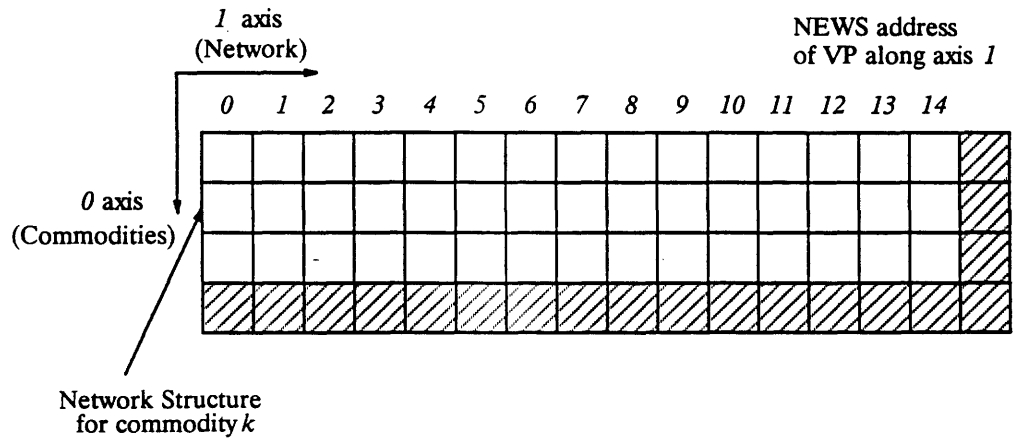
2. Absolute error on network equality constraints for minor iterations:

$$\max_{i\in V_O, j\in V_D, k\in<K>} \left\{ \mid s_k(i) - \sum_{j\in\delta^+(i)} x_k(i,j) \mid , \mid d_k(j) - \sum_{i\in\delta^-(j)} x_k(i,j) \mid \right\} \leq \epsilon_2 \tag{151}$$

   In all experiments we set $\epsilon_2 = 10^{-4}$.

In this section we provide a summary of computational results in order to highlight certain aspects of the performance of the algorithm and illustrate its suitability for the solution of very large problems. The program was compiled on a SUN 4/280 FE using compiler flags -O -cm2. We used in all runs a CM–2 with 32-bit floating point accelerators at Thinking Machines Corporation. All times are in seconds. Data input/output is excluded, but time for the transfer of data between the FE and the CM is included, together with all time spent in communications on the CM. All times reported are in total CPU time as recorded by the FE. This includes CPU time for execution of the C/Paris code and time





Data Fields in the k–th row corresponding to commodity k.

NEWS address of VP along axis 1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Node	1	1	1	2	2	3	3	3	3	4	4	4	5	5	5
Segment bits	1	0	0	1	0	1	0	0	0	1	0	0	1	0	0
Supply Data	$s_k(1)$	$s_k(1)$	$s_k(1)$	$s_k(2)$	$s_k(2)$	$s_k(3)$	$s_k(3)$	$s_k(3)$	$s_k(3)$	$s_k(4)$	$s_k(4)$	$s_k(4)$	$s_k(5)$	$s_k(5)$	$s_k(5)$
Demand Data	$d_k(1)$	$d_k(1)$	$d_k(1)$	$dx(2)$	$d_k(2)$	$d_k(3)$	$d_k(3)$	$d_k(3)$	$d_k(3)$	$d_k(4)$	$d_k(4)$	$d_k(4)$	$d_k(5)$	$d_k(5)$	$d_k(5)$
Joint Capacity	∞	$U(1,3)$	$U(1,4)$	∞	$U(2,3)$	∞	$U(1,3)$	$U(2,3)$	$U(3,5)$	∞	$U(1,4)$	$U(4,5)$	∞	$U(3,5)$	$U(4,5)$
Send address in NEWS coordinates along axis 1	0	6	10	3	7	5	1	4	13	9	2	14	12	8	11

Figure 2: Representing sparse multicommodity networks on the CM.

for the controlling program and FE calculations. Most runs were carried out on a lightly used FE and the CM times consume more than 95% (up to 99% in some cases) of the total time.

The dense implementation runs at approximately 1.6 GFLOPS when implemented in C/Paris. It is possible to design an alternative implementation based on the NEWS grid, that minimizes the amount of communication required. Such an implementation was carried out in CMIS and is described in McKenna and Zenios [1990]. It was observed to achieve 3 GFLOPS rate in solving single commodity problems. Unless otherwise stated all subsequent experiments are using the C/Paris implementation.

4.1 Problem generator

We wrote a problem generator that provides control over several characteristics of the test problems. The generator was also written in C/Paris on the CM. It is thus possible to generate extremely large problems in memory and pass them on to the solver without the need to transfer data to an external storage device — a task that would take several hours for the bigger problems. The input parameters for the generator are: (1) Number of origin and destination nodes m_O and m_D respectively, (2) Number of commodities K, (3) Condition number ρ, (4) Largest coefficient for linear term, max-c, (5) Percentage of joint capacity constraints that are active at the optimal solution α, (6) The tightness of the active joint capacity constraints β, (7) Maximum supply or demand at each node, max-sd. The generator accepts as input the seven control parameters and generates a problem in three steps:

Step 1: Generate K single commodity problems: Set up a two-dimensional NEWS grid of $\lceil m_O \rceil_2 \times \lceil m_D \rceil_2$ active VPs. Generate the quadratic objective function coefficients $w_k(i,j)$ in the range [1,ρ] using a uniform random distribution. Generate the linear objective function coefficients $c_k(i,j)$ in the range [1, max-c]. Generate supply and demand values for origin and destination nodes respectively in the range [1, max-sd]. Scale all supply values by $\frac{\sum_{j=1}^{m_D} d_k(j)}{\sum_{i=1}^{m_O} s_k(i)}$ to ensure that the equality constraints are feasible.

Step 2: Solve the K uncapacitated, single commodity problems generated in Step 1. (We use the algorithm of Zenios and Censor [1990].)

Step 3: Calculate the sum of the optimal flows from Step 2 over all commodities for each arc, say $ex(i,j)$. Choose, at random, α% of the arcs that will have active GUB

constraints. For the arcs so chosen, set $U(i,j) = \beta \cdot ex(i,j)$. The rest of the arcs are virtually unrestricted, and the joint capacity constraints are set to $U(i,j) = \frac{2}{\beta} \cdot ex(i,j)$.

At this point any arcs that have $ex(i,j) = 0$ are removed from the problem. In general, we found that leaving these arcs in the problem with a large upper bound would make the test problems significantly easier. Some of the problems solved at the early stages of our experimentation kept all the arcs in the problem. Whenever, in subsequent sections, the number of arcs is precisely $m_O \times m_D$, then the problem is relatively easy.

Subsequent sections will describe in detail the parameters used to generate each problem. When no such information is given, then the following base-case test problem is being used:
Nodes: $m_O = m_D = 256$
Commodities: K=5
Condition number: $\rho = 10^2$
Largest coefficient for linear term: max-c=100
Percentage of active joint capacity constraints: $\alpha = 10\%$
Tightness of active joint capacity constraints: $\beta = 0.90$
Maximum supply or demand: max-sd=100.

When information is provided only for some of the input parameters, then the remaining parameters have the values given above for the base case.

4.2 Algorithmic performance

The algorithm is a first-order method, and as such it is expected to have a tailing effect. A small test problem is used to illustrate the performance of the algorithm. Figure 3 plots the absolute maximum error of the GUB constraints $\left(max_{(i,j)\in\mathcal{E}} \left\{0, \sum_{k=1}^{k} x_k(i,j) - U(i,j)\right\}\right)$ at successive major iterations. Within each major iteration the figure illustrates the absolute error of the equality constraints at selected minor iterations. Following a major iteration, the joint capacity constraint error is zero (recall that Step 2 of the algorithm will satisfy exactly the joint capacity constraints), but the error of the equality constraints is large. Successive minor iterations reduce the error at the equations and increasingly violate the joint capacity constraints. Depending on which constraints are "soft" in a given model (i.e., the GUB constraints or the equality constraints), the results of Figure 3 provide some guidance on when should the algorithm be terminated.

The question is raised whether the minor iterations should be terminated with a loose tolerance at the early major iterations. For the sparse implementation, such a strategy is not advisable for the following reason: In order to execute the minor iterations, each

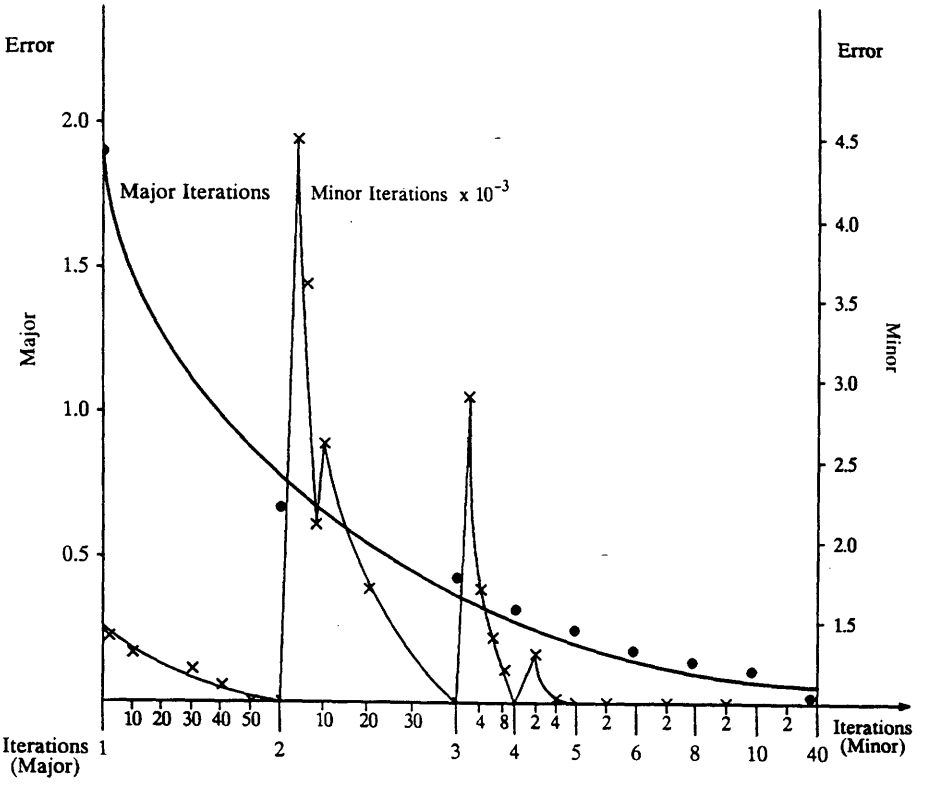


Figure 3: Maximum error at successive major and minor iterations.

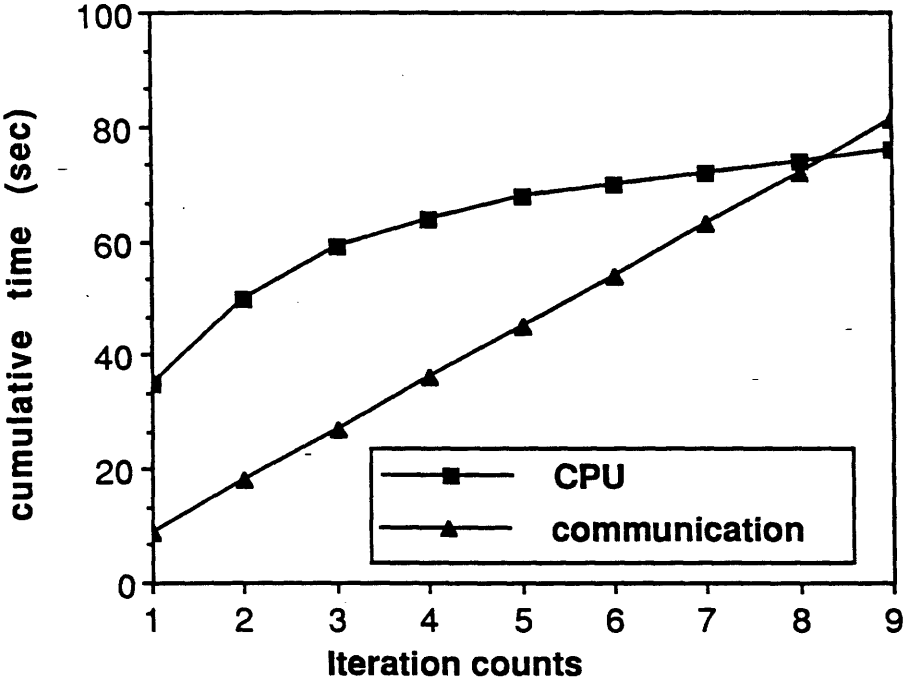


Figure 4: Computation and communication times for the dense implementation.

commodity has to be loaded into the CM memory from the FE. A significant amount of time is consumed in this step. As illustrated in Figure 4, after the first few major iterations more time is spent in transferring data from the FE to the CM than in computations. Terminating the algorithm with looser minor iteration tolerance will increase the number of major iterations, and hence the communication time. After the first 1–2 major iterations, the decrease in computing time will not compensate for the increase of communication time. The numerical values from Figure 4 on which this claim is based are the following:

Major itn.	Computation time	Communication time
0	447.0	20.0
1	35.0	9.2
2	15.5	8.9
3	9.4	9.2
4	5.6	8.6
5	3.8	8.6
6	2.1	9.4
7	2.2	8.7

This is one example where a strategy that would be efficient on a serial computer, or a coarse-grain parallel computer, is not advisable on a massively parallel system. The time spent in communications is critical in the choice of internal algorithmic tactics.

4.3 The effects of problem structure

It is well established in the optimization folklore that the performance of a nonlinear programming algorithm may differ significantly for different problem characteristics. In this section we look at the performance of the quadratic programming algorithm when the following problem parameters change: (1) Number of commodities K, (2) Condition number ρ, (3) Percentage of active GUB constraints, α, and (4) Tightness of active GUB constraints, β.

4.3.1 Increasing number of commodities

Figure 5 illustrates both solution time and number of major iterations as the number of commodities increases for a given problem size. Problems are getting only slightly more difficult with increasing number of commodities, as manifested by the small increase in the number of major iterations. However, solution times increase linearly with the number of commodities. This observation provides empirical verification that the amount of com-

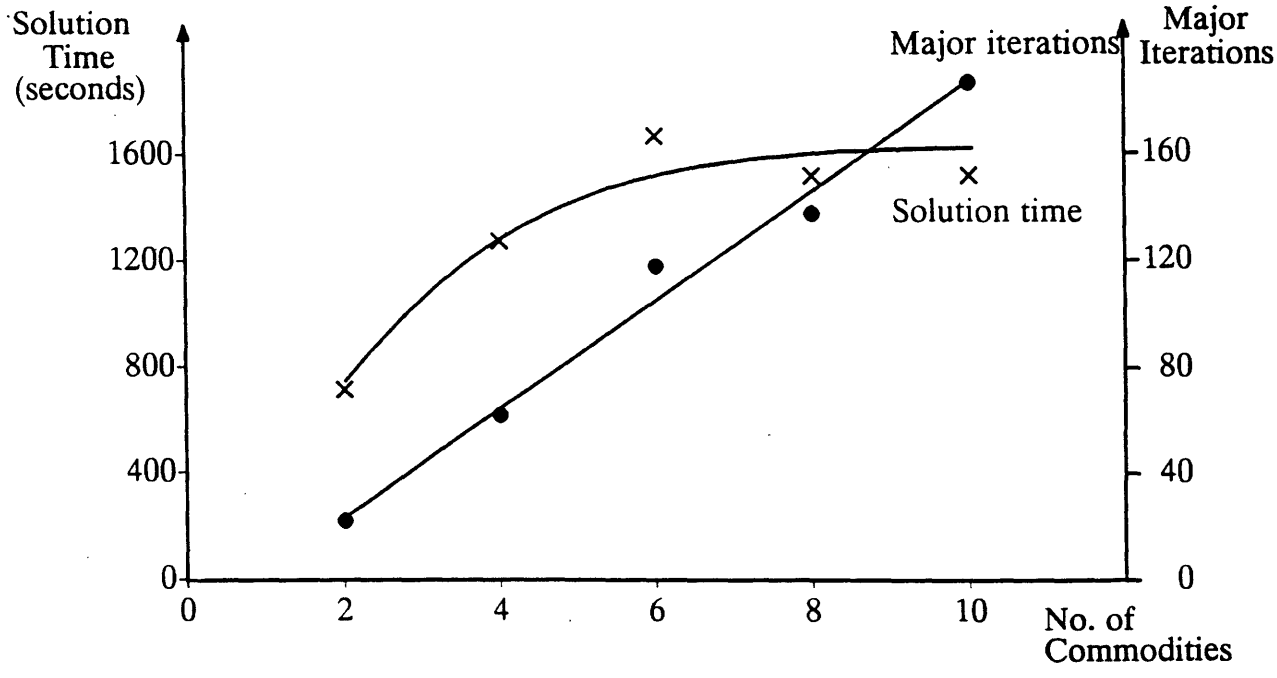


Figure 5: Effect of increasing the number of commodities K on the performance of the algorithm.

putation and communication required per iteration is a linear function of the number of commodities.

There is an interesting implication from the results of this figure: If a massively parallel architecture scales linearly in size with the number of commodities, then larger problems can be solved with only slight increase of solution time. This increase will be proportional to the increase in major iterations.

4.3.2 Increasing condition number

First order algorithms, like those presented here, are very sensitive to ill-conditioning. Figure 6 illustrates the performance of the algorithm as the condition number of the problem increases, and for problems with varying number of commodities. The adverse impact of ill-conditioning is more significant for problems with more commodities. For condition number $\rho = 10^2$, the algorithm can solve problems with a large number of commodities. (In Section 4.5, we report solution profiles for problems with up to 20 commodities.) For condition number $\rho = 10^3$, the algorithm can still solve problems with many commodities, but at significant increase in computing time. For condition number $\rho = 10^4$, the algorithm has a very slow rate of convergence when applied to problems with more than two commodities. The information in this section provides general guidelines on the potential difficulty of a

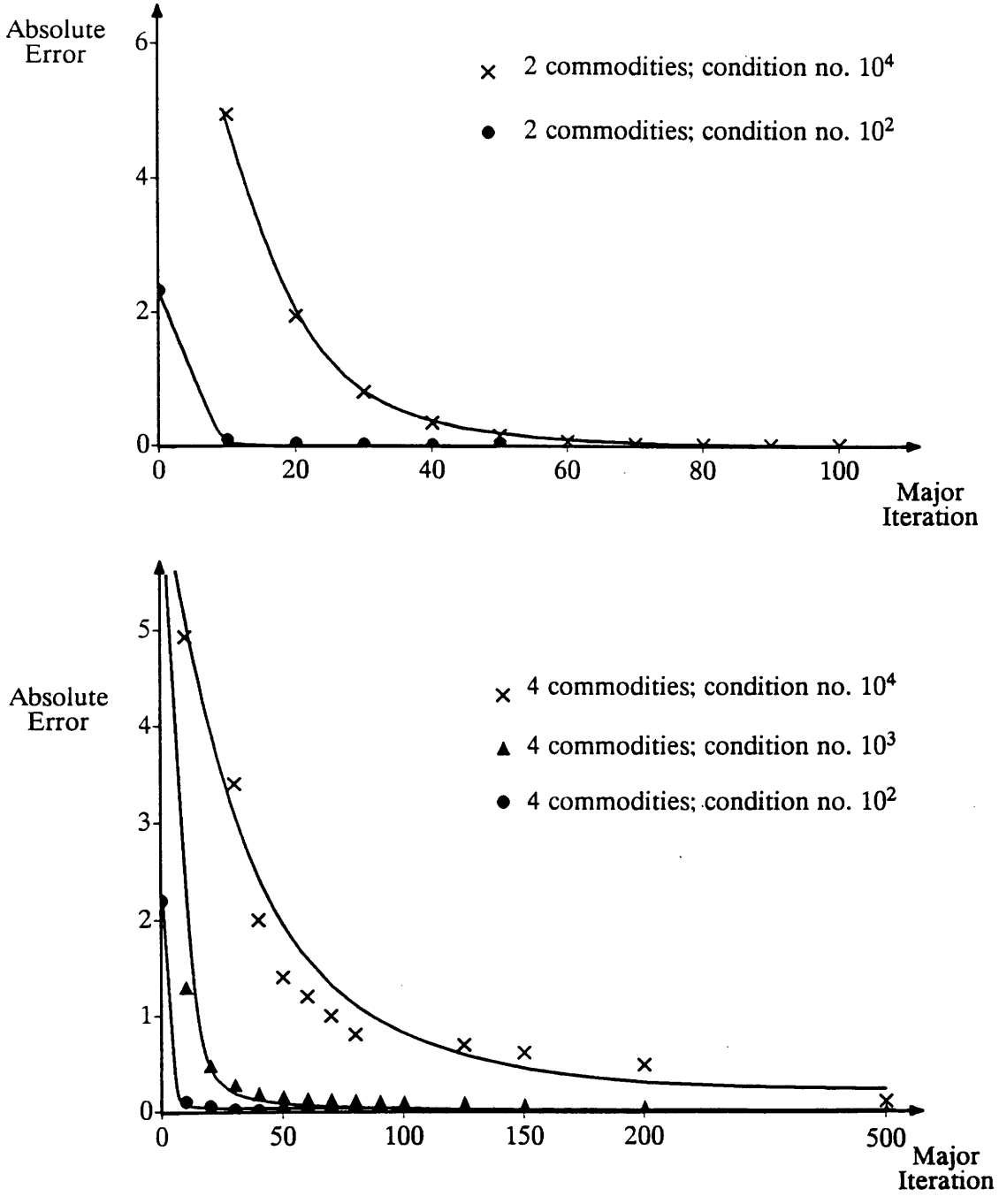


Figure 6: Effect of increasing the condition number ρ on the performance of the algorithm.

given test problem.

4.3.3 Increasing percentage of active GUB constraints, α.

As the number of joint capacity constraints that are active at the optimal solution increases, the problems become more difficult. Figure 7 shows the increase in: (1) Major iterations, (2) Total solution time, and (3) Solution time per major iteration, as the percentage of active constraints α increases.

We observe that the increase in solution time is primarily due to the increase in number of major iterations. Hence, no improvements in performance can be anticipated with increases in the size of the computer. It is also usually unknown a priori how many constraints are active at the solution. Hence, the results of this section are of limited practical value. They mainly illustrate two facts. First, the algorithm is capable of solving problems with a significant number of active GUB constraints, and second, the solution time increases only linearly with increase in the number of active constraints.

4.3.4 Increasing tightness of active GUB constraints, β.

Intuitively, one expects the following: If the joint capacity constraint is larger than the sum of the unconstrained optimal flows on all commodities, then the problem is trivial. It will be solved in one major iteration. If the joint capacity is slightly less than the sum of unconstrained flows, then some flow can be diverted easily to adjacent arcs. If the joint capacity is significantly less than the sum of unconstrained flows, then more flow has to be diverted to adjacent arcs and the problem gets more difficult.

This intuition is confirmed in Figure 8 when the tightness of the active capacity constraints ranges from $\beta = 0.97$ to $\beta = 0.90$. (Recall that for $\beta = 1.00$ the joint capacity constraints are not active, and they become more tight as β decreases.) From the same figure, however, it appears that the problems get marginally more difficult as β decreases to 0.80. Additional experimentation is needed before one attempts to devise an explanation.

4.4 The Performance of the Sparse Implementation

Table 1 summarizes the results in solving identical problems using both the dense and sparse implementations. All of the test problems solved here are totally dense, and hence this comparison is biased against the sparse implementation. Nevertheless, some interesting observations can be made which are very encouraging for the performance of the sparse implementation. If the computer configuration does not have a sufficient number of pro-

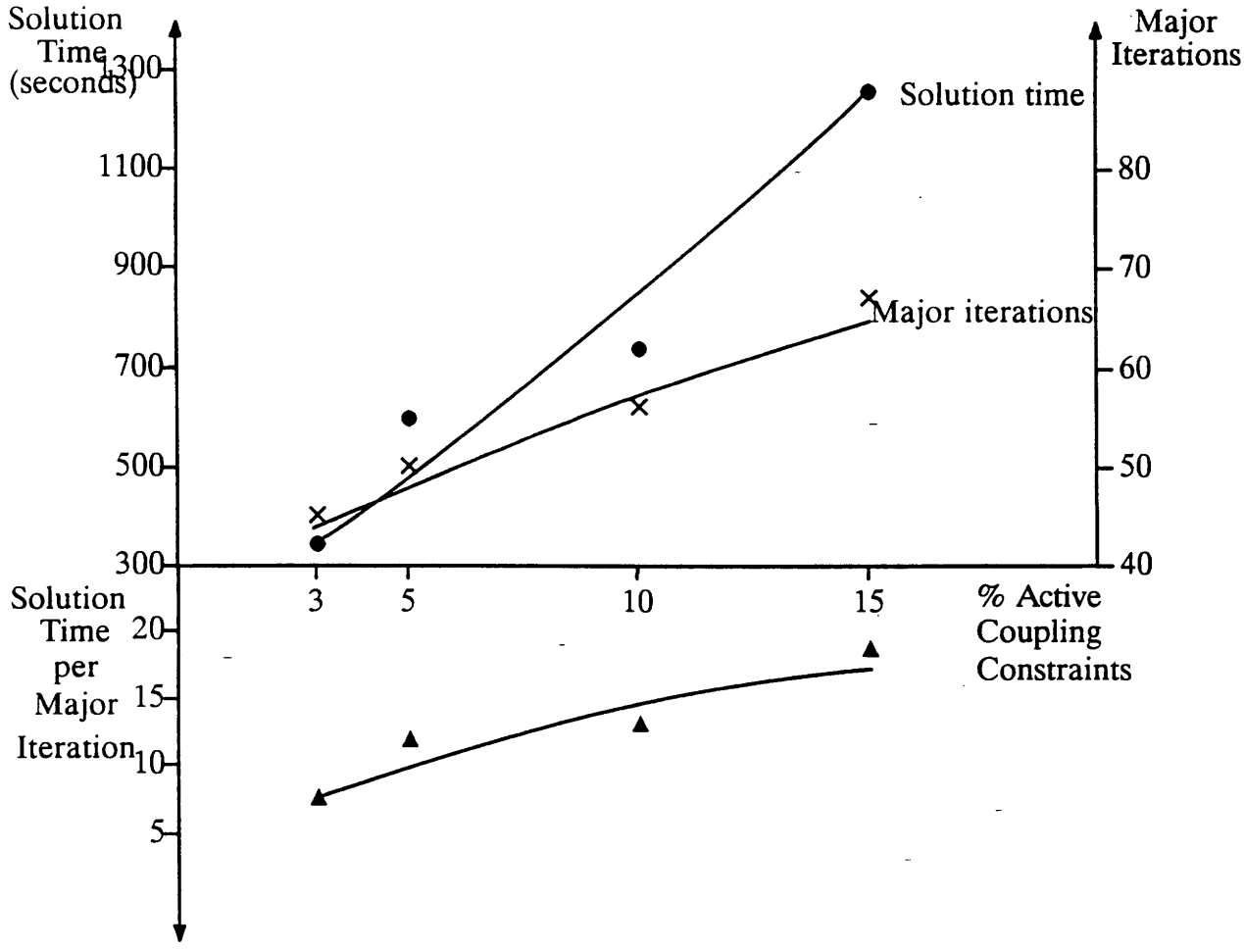


Figure 7: Effect of increasing the percentage of active GUB constraints α on the performance of the algorithm.

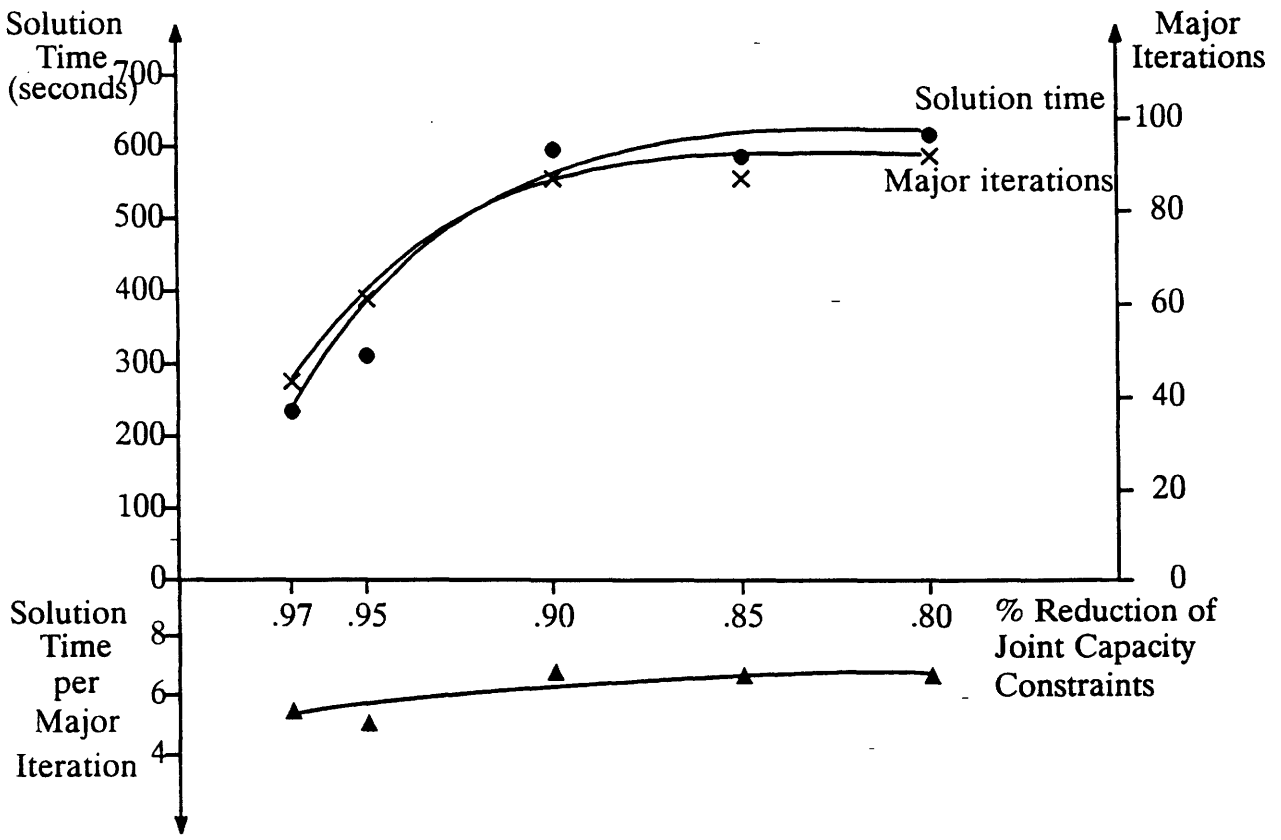


Figure 8: Effect of increasing the tightness of the joint capacity constraints β on the performance of the algorithm.

No.	Problem size	Solution characteristics	4K CM–2		8K CM–2
			Dense Implementation	Sparse Implementation	Sparse Implementation
1	8 com. 128 nodes 4096 arcs	VP ratio CM time(sec)	1 144	32 305	16 169
2	8 com. 64 nodes 1024 arcs	VP ratio CM time(sec)	1 130	8 218	4 139
3	8 com. 32 nodes 256 arcs	VP ratio CM time(sec)	1 156	1 25	NA

Table 1: Comparing the dense and sparse implementations. (Solution times in seconds.)

cessors to store all the commodities simultaneously in their sparse representation, then the dense implementation is superior. The time spent in transferring data from the FE to the CM is compensated by the improved computing performance achieved when the algorithm executes at a low VP ratio. Compare the solution time of problems no. 1 and 2 on the 4K CM–2, where the dense implementation is faster. However, the sparse implementation can run faster on a bigger machine, while the dense implementation is already executed at a VP ration of 1. Compare the solution times for problems no. 1 and 2 with the dense implementation running at a VP ratio=1 on a 4K CM–2, and the sparse implementation running on an 8K CM–2 with VP ratios 16 and 4 respectively. The sparse implementation is at par with the dense implementation and could improve even further with an increase in the number of processing elements. When both the dense and sparse implementation run with VP ratio 1, then the sparse implementation can be significantly faster — see the results for problem no. 3.

Identifying the precise conditions under which the sparse implementation should be preferred is a complex problem. A theoretical analysis of this problem, and empirical verifications, are the topic of a current study. We discuss here the factors that enter into the analysis:

1. The dense implementation runs at peak rate of 3 GFLOPS when solving dense problems on a 64K CM–2. This computing rate decreases linearly with increase in the sparsity of the problem, and decreases almost linearly with decrease in the number

of PEs. The sparse code runs at 140 MFLOPS. Both codes achieve these rates with high VP ratios — a small loss in computing rate is observed with lower VP ratios.

2. The dense code solves the commodities one at a time in K serial steps. The sparse code solves all K commodities simultaneously.

3. The dense code executes at a VP ratio $\frac{\lceil m_O \rceil_2 \lceil m_D \rceil_2}{P}$. The sparse code executes at a VP ratio $\frac{\lceil K(2n+(m_O+m_D)) \rceil_2}{P}$.

4. The dense implementation requires a communication step between the FE and the CM for each commodity.

The following examples illustrate how these factors affect the performance of the algorithm, and could provide an explanation for the results of Table 1.

Example 1: Consider the solution time for problem no. 3 that was solved under VP ratio=1 for both dense and sparse implementations. The dense code executes at $\frac{3GFLOPS}{64/4} = 187.5$ MFLOPS on the 4K CM–2. The sparse code runs at 140 MFLOPS. Hence, problem no. 3 could be solved in $156 \times \frac{187.5}{140} = 208.93$ sec if the sparse code were used to solve the 8 commodities one at a time. If all 8 commodities are solved in parallel, the solution time would be $\frac{208.93}{8} = 26.1$ sec. This estimate is slightly higher than the 25 sec solution time observed by the sparse implementation, since it also includes time spent in transferring the commodities from the front-end to the CM at each major iteration of the dense code. For the sparse code, this transfer of data is executed only once.

Example 2: Repeating the same line of reasoning for problem no. 2 results in estimated times that are significantly different from the observed time for the sparse code. The solution time would increase to $130 \times \frac{187.5}{140} = 174.1$ sec, using the sparse code to solve one commodity at a time. Solving all 8 commodities in parallel, assuming we could still run at VP ratio=1, completes in 21.8 sec. At a VP ratio=4, the solution time would be $\approx$ 87 sec. At a VP ratio=8, the estimated solution time would be $\approx$ 174 which is much lower than the observed 218 sec. Clearly, the arguments in Example 1 do not capture the complex interactions among the several components of the two algorithms, when each is executed at different VP ratios.

4.5 Solving large scale problems

As a final exercise, we generated and solved some very large problems. The results are reported in Table 2. The algorithm achieves a good level of accuracy in number of major

iterations that range from 20–100. The largest problems (no. 5–10) have more than 1 million variables and 10 thousand equations and they are solved in well within 1 hour of wall clock time. The same problem could be solved in less than 10 minutes on a 32K CM–2 and less than 5 minutes on a fully configured 64K CM–2.

5 Concluding Remarks

This paper has developed a class of algorithms for nonlinear multicommodity transportation problems. The algorithms induce a fine-grain decomposition of the problem: First, by node for each commodity, and then by arc. As a result, it is possible to implement these algorithms on massively parallel computer architectures. Of course, implementations on small-scale parallel machines are also possible, although such implementations would not exploit fully the potential of the algorithms.

The algorithms appear effective in solving problems of medium difficulty (condition numbers 10-1000) to a good level of accuracy. Nevertheless, as first order methods, they could be severely affected by ill-conditioned problems. Also, the attainment of very high accuracy could come with significant increase in the number of iterations and computer times.

Massively parallel implementations on the Connection Machine CM–2 have been possible even for sparse problems. The implementations are very efficient, and match the parallel nature of the algorithms with the computer architecture of the CM–2. We expect significant improvements in the performance of massively parallel architectures over the coming years. For example, the current version of the CM–2 has processing elements that operate at 7MHZ; this is significantly lower than the operating cycle of personal computers. Also, 64-bit WEITEK floating-point accelerators are now being added to some configurations. The algorithms developed here could then achieve higher accuracy with little additional effort. The current model of C/Paris — based on field-wise representation of data — is far from being the most efficient one for the CM–2. The optimization algorithms developed here should be able to benefit from improvements in the computer models without additional effort on our part.

It is a severe limitation that these algorithm are applicable only to nonlinear problems. Nevertheless, they are the building blocks for solving linear programs: The quadratic optimization algorithms in the context of proximal minimization of Rockafellar [1976] and the entropy optimization algorithms in the context of the framework developed by Censor and Zenios [1990]. This is the topic of a current study.

No.	Problem size		Cond. No.	Major Itns.	Error		Time	
	Network formulation	Lin. prog. formulation			% GUB const.	Absolute node const.	4K CM–2	32K CM–2
1	2 com. 256 nodes 12976 arcs	512 eqns. 1672 GUB 25952 vars.	10^4	90	.09	10^{-5}	20:00	3:10
2	2 com. 256 nodes 15276 arcs	512 eqns. 1675 GUB 30534 vars.	10	100	.11	10^{-7}	1:00	0:10
3	5 com. 512 nodes 16370 arcs	1280 eqns. 1634 GUB 83850 vars.	10^2	86	.09	10^{-6}	1:50	:20
4	5 com. 512 nodes 65478 arcs	2560 eqns. 6552 GUB 327435 vars.	10^2	NA	.09	10^{-6}	4:30	0:45
5	5 com. 1024 nodes 262000 arcs	5120 eqns. 26181 GUB 1310000 vars.	10^2	41	.09	10^{-6}	14:10	2:15
6	10 com. 1024 nodes 262144 arcs	10240 eqns. 25957 GUB 2621140 vars.	10^2	37	.09	10^{-6}	31:30	4:55
7	20 com. 1024 nodes 262144 arcs	20480 eqns. 26260 GUB 5424880 vars.	10^2	90	.09	10^{-5}	45:00	7:20
8	2 com. 2048 nodes 1002338 arcs	4096 eqns. 104443 GUB 2004678 arcs.	10^2	17	.08	10^{-6}	14:35	2:15
9	5 com. 2048 nodes 1048074 arcs	10240 eqns. 104088 GUB 5240370 vars.	10^2	19	.07	10^{-6}	40:40	6:20
10	8 com. 2048 nodes 1048570 arcs	16384 eqns. 104521 GUB 8384560 vars.	10^2	22	.09	10^{-6}	29:50	4:40

Note: Time on the 32K CM–2 is estimated, dividing the time on the 4K CM–2 by 6.4. This is the empirically observed improvement in performance as the VP ratio is reduced by a factor of 8, when moving from the 4K to the 32K CM–2.

Table 2: Solving large scale problems. (Time in min:sec).

As a postscript, we add that massively parallel algorithms — like those proposed here — have been developed by Nielsen and Zenios [1990] for stochastic programming problems with network recourse. Preliminary implementations on the Connection Machine CM–2 for some financial modeling applications reinforce the encouraging results reported here for multicommodity flows.

Acknowledgements: Professor Yair Censor deserves honorary co-authorship on this paper for numerous illuminating discussions on row-action algorithms and his constant encouragement. I also benefited from the comments of J. Mesirov and S. Nielsen. It is a pleasure to acknowledge that the research reported here was completed while I was with Thinking Machines Corporation and the Operations Research Center at M.I.T.. Partial support has been provided by NSF grant CCR-8811135 and AFOSR grant 89-0145. Computing resources were made available by the North-east Parallel Architectures Center (NPAC) of Syracuse University, NY.

References

[1] A.A. Assad. Multicommodity network flows - a survey. *Networks*, 8:37–91, 1978.

[2] D.P. Bertsekas. Algorithms for nonlinear multicommodity network flow problems. In A. Bensoussan and J.L. Lions, editors, *International Symposium on Systems Optimization and Analysis*, pages 210–224, Springer-Verlag, 1979.

[3] A. Bjorck and T. Elfving. Accelerated projection methods for computing pseudo-inverse solutions of systems of linear equations. *BIT*, 19:145–163, 1979.

[4] G.E. Blelloch. *Vector Models for Data-Parallel Computing.* The MIT Press, Cambridge, Massachusetts, 1990.

[5] R. Bramley and A. Sameh. *Row projection methods for large nonsymmetric linear systems.* CSRD Report 957, University of Illinois, Champaign-Urbana, Jan. 1990.

[6] Y. Censor. Parallel application of block-iterative methods in medical imaging and radiation therapy. *Mathematical Programming, Series B*, 42(2):307–326, 1988.

[7] Y. Censor. Row-action methods for huge and sparse systems and their applications. *SIAM Review*, 23:444–464, 1981.

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

[9] 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*, pages 147–165, Banach Center Publications, PWN - Polish Scientific Publisher, Warsaw, Poland, 1989.

[10] Y. Censor and S.A. Zenios. Interval constrained matrix balancing. *Linear Algebra and its Applications*, 1989a. (to appear).

[11] Y. Censor and S.A. Zenios. *The Proximal Minimization Algorithm with D-functions.* Working paper, Decision Sciences Department, The Wharton School, University of Pennsylvania, Philadelphia, PA, 1989b.

[12] R.J. Chen and R.R. Meyer. Parallel optimization for traffic assignment. *Mathematical Programming, Series B*, 42(2), 1988.

[13] J. Eckstein. *Implementing and Running the Alternating Step Method on the Connection Machine CM–2.* Working paper 91-005, Division of Research, Harvard Business School, Boston, MA, 1990.

[14] T. Elfving. An algorithm for maximum entropy image reconstruction from noisy data. *Mathematical Computer Modeling*, 12:729–745, 1989.

[15] R. Gallager. A minimum delay routing algorithm using distributed computation. *IEEE Transactions on Communication*, COM-25:73–85, 1977.

[16] M. Gondran and M. Minoux. *Graphs and Algorithms.* John, Wiley and Sons, N.Y., 1984.

[17] G.T. Herman. *Image Reconstruction from Projections: The Fundamentals of Computerized Tomography.* Academic Press, New York, 1980.

[18] W. D. Hillis. *The Connection Machine.* The MIT Press, Cambridge, Massachusetts, 1985.

[19] J.L. Kennington. A survey of linear cost multicommodity network flows. *Operations Research*, 26:209–236, 1978.

[20] M. McKenna and S.A. Zenios. An optimal parallel implementation of a quadratic transportation algorithm. In *SIAM Conference on Parallel Processing for Scientific Computing*, 1990. (to appear).

[21] S. Nielsen and S.A. Zenios. *Massively Parallel Algorithms for Nonlinear Stochastic Network Problems.* Working paper, Decision Sciences Department, The Wharton School, University of Pennsylvania, Philadelphia, PA, 1990a.

[22] S. Nielsen and S.A. Zenios. *Sparse vs Dense Implementations of Network Problems on the Connection Machine.* Working paper, Decision Sciences Department, The Wharton School, University of Pennsylvania, Philadelphia, PA, 1990b.

[23] A.R. De Pierro and A.N. Iusem. A relaxed version of bregman's method for convex programming. *Journal of Optimization Theory and its Applications*, 5:421–440, 1986.

[24] R. T. Rockafellar. Augmented lagrangians and applications to proximal point algorithms in convex programming. *Mathematics of Operations Research*, 1:97–116, 1976.

[25] G.L. Schultz and R.R. Meyer. *A Structured Interior Point Method.* Working paper, Computer Science Department, The University of Wisconsin-Madison, Madison, WI, 1990.

[26] R. D. Wollmer. Multicommodity networks with resource constraints: the generalized multicommodity flow problem. *Networks*, 1:245–263, 1972.

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

[28] S.A. Zenios and Y. Censor. *Massively Parallel Row-Action Algorithms for Some Nonlinear Transportation Problems.* Report 89–09–10, Decision Sciences Department, The Wharton School, University of Pennsylvania, Philadelphia, PA, 1989.

[29] S.A. Zenios and Y. Censor. *Vector and Parallel Computing with Block-iterative Algorithms for Medical Image Reconstruction.* Report 88–09–10, Decision Sciences Department, The Wharton School, University of Pennsylvania, Philadelphia, PA, 1988.

[30] S.A. Zenios and S. Nielsen. *Massively Parallel Algorithms for Singly Constrained Nonlinear Programs.* Report 90–03–01, Decision Sciences Department, The Wharton School, University of Pennsylvania, Philadelphia, PA, 1990.

[31] S.A. Zenios, M.C. Pinar, and R.S. Dembo. *Linear-Quadratic Penalty Forms for the Decomposition of Network Structured Problems.* Working paper, Decision Sciences Department, The Wharton School, University of Pennsylvania, Philadelphia, PA, 1990.