A MASSIVELY PARALLEL ALGORITHM FOR NONLINEAR STOCHASTIC NETWORK PROBLEMS

by

S. S. Nielsen and S. A. Zenios

OR 237-90 November 1990

A Massively Parallel Algorithm for Nonlinear Stochastic Network Problems

Soren S. Nielsen
Stavros A. Zenios

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

October 1990

Abstract

We develop an algorithm for solving nonlinear two-stage stochastic problems with network recourse. The algorithm is based on the framework of *row-action* methods. The problem is formulated by replicating the first-stage variables and then adding *non-anticipativity* side constraints. A series of (independent) deterministic network problems are solved at each step of the algorithm, followed by an iterative step over the non-anticipativity constraints. The solution point of the iterates over the non-anticipativity constraints can be obtained analytically. The row-action nature of the algorithm makes it suitable for parallel implementations.

A data representation of the problem is developed that permits the massively parallel solution of all the scenario subproblems concurrently. The algorithm is implemented on a Connection Machine CM-2 with up to 32K processing elements and achieves computing rates of 250 MFLOPS. Very large problems — 8192 scenarios with a deterministic equivalent nonlinear program with 1,272,160 variables and 495,616 constraints — are solved within a few minutes. We report extensive numerical results regarding the effects of stochasticity on the efficiency of the algorithm.

Contents

1 Introduction

It has long been recognized (Dantzig [1955]) that traditional deterministic mathematical programs are not suitable for capturing the truly dynamic behavior of most real-world applications. A primary reason is that such applications involve data uncertainties. Data uncertainties arise in a dynamic setting because information which will be needed in subsequent decision stages is not yet available to the decision maker or modeler. In applications from financial planning such information would be future asset prices and returns, or uncertain future liabilities in insurance and pension fund management. Applications from logistics planning and utility service scheduling exhibit uncertain supplies or demands, as well as uncertain transportation or construction costs.

Stochastic programming was first proposed (independently by Dantzig [1955] and Beale [1955]) as a remedy for these problems. Although there has been significant progress in the ability of researchers to solve general stochastic programs, these programs often turn out to be very complex for practical applications. Their exact solution on the computer often proves prohibitively expensive. Solving deterministic "mean value" or "worst case" approximations may lead to solutions which are far from optimal, Birge [1982]. Reformulating the stochastic program into a large-scale deterministic equivalent taking uncertainties explicitly into account is often possible. However, the deterministic program may again be very expensive to solve. This is not only because of sheer size, but also because any special structure exhibited by the stochastic program is largely lost in the deterministic equivalent.

Research over the last twenty years has largely concentrated on (i) devising efficient decomposition methods for solving the deterministic equivalent program (Dantzig, Dempster and Kallio [1981], Van Slyke and Wets [1966, 1969], Ruszczynski [1986], Infanger [1990]), (ii) designing or exploiting parallel computing machinery to speed up the solution process (Wets [1985], Ruszczynski [1988], Dantzig [1985]), and (iii) on retaining any special structure present in the stochastic program, Qi [1985], Wallace [1984, 1986], Mulvey and Vladimirou [1988, 1989].

The problem can be solved with methods based on Benders' decomposition (Infanger [1990]). Van Slyke and Wets [1966, 1969] proposed an "L-shaped" decomposition procedure based on adding feasibility and optimality cuts to a master problem until convergence. Ruszczynski [1986] developed a regularized version of this algorithm. None of these decomposition algorithms retain the structure of the original program during the solution process.

More recently, Rockafellar and Wets [1987], (see also Wets [1988]) introduced the *progressive hedging* algorithm, a decomposition algorithm based on Augmented Lagrangean theory. This algorithm retains any structure present in the sub-problems and furthermore is suitable for coarse-grained parallel solution. Is was used by Mulvey and Vladimirou [1989] as the basis for a distributed algorithm for solving stochastic programs with network recourse.

Wets [1985] proposed the use of a large number of very simple processors to solve (multistage) stochastic programs. Since then, multiprocessor systems have become widely

available, causing wide-spread interest in the development and implementation of parallel algorithms for stochastic programs. Ruszczynski [1988] develops a new decomposition method for (multistage) stochastic programs, which involves (possibly) parallel solution of the subproblems. Ariyawansa and Hudson [1989] implement and test on a Sequent/Balance multiprocessor a version of Van Slyke and Wets' [1969] algorithm. Dantzig and Glynn [1990] demonstrate that the combination of nested decomposition, Monte Carlo importance sampling and parallel programming can be combined to solve a class of multistage stochastic programs.

In this paper we consider the application of the *row-action algorithm* of Censor and Lent [1981] in the development of a fine-grain parallel algorithm for (strictly convex) stochastic network problems. Our motivations are the recognition that a great number of important stochastic problems have an inherent network structure, Mulvey and Vladimirou [1988], the existense of decomposition methods which retain the network structure in the resulting subproblems (the progressive hedging algorithm of Rockafellar and Wets [1988] and the row-action algorithm we propose in the present study), and the recent advances in employing massively parallel computers to solve extermely large network problems, Zenios and Lasken [1988] and Zenios and Nielsen [1990].

We employ a decomposition technique for two-stage generalized stochastic network problems by "splitting" (or replicating) first-stage decision variables. This reformulation preserves the network structure. The subsequent application of a row-action algorithm results in a decomposition scheme that is suitable for both coarse-grain and fine-grain parallel computing. The algorithm is implemented on a massively parallel Connection Machine CM–2 with up to 32K processing elements. We extend the sparse data structure of Zenios and Lasken [1988] to represent the stochastic networks and allow concurrent solution of all the subproblems corresponding to individual scenarios. We report on the solution of medium-sized applications in a few seconds, and on the solution of problems with 8192 scenarios in a few minutes. We also report on the results of numerical experiments on the effect of stochasticity on the performance of the algorithm.

This paper is the third in a series that deals with the massively parallel solution of problems with network structures. Zenios and Censor [1989] studied the nonlinear transportation problems, and algorithms for multi-commodity problems are studied in Zenios [1990].

The rest of the paper is organized as follows: In Section 2 we present the row-action algorithm and its specialization for stochastic, generalized network problems. Section 3 covers the Connection Machine CM–2 and the implementation of the algorithm, including the concurrent solution of the scenario subproblems. Section 4 reports on numerical results for a number of test problems. The experiments establish the suitability of the algorithm for solving very large stochastic network problems, and also indicate conditions under which the algorithm might exhibit poor convergence.

1.1 Problem Formulation

We now formally define the two-stage stochastic nonlinear problem with recourse. The dynamics of the situation we are modeling are as follows:

> A decision maker must make a decision regarding current actions, facing an uncertain future. After these *first-stage* decisions are made, a realization of the uncertain future is observed, and the decision maker determines an optimal *second-stage* decision. The objective is to minimize the total expected cost or to maximize expected final wealth or utility of final wealth.

This framework (and our algorithmic approach introduced next) can be generalized to more than two stages. In this study, we concentrate on the two-stage variant.

1.1.1 The Two-stage Stochastic Program

Let $< n >$ denote the set $\{1, 2, ..., n\}$ and x^T denote the transpose of the vector x. $x^T y$ denotes the inner product of two real vectors x and y. Bold letters are used to denote stochastic quantities, and the corresponding roman letters designate instances of the stochastic quantities.

The two-stage nonlinear stochastic programming problem can be formulated as follows:

$$\begin{aligned}
\textbf{[SNLP]} \qquad & \underset{x \in \Re^{n_1}}{\text{Minimize}} \quad f(x) + \mathcal{Q}(x) && (1)\\
& \text{Subject to} \quad Ax \quad = b && (2)\\
& \qquad\qquad\quad 0 \leq x \leq u^x && (3)
\end{aligned}$$

where

$$\mathcal{Q}(x) = \mathrm{E}\{Q(\mathbf{g}, \mathbf{r}, \mathbf{B}, \mathbf{v} \mid x)\}$$

and

$$\begin{aligned}
Q(g, r, B, v \mid x) = \quad & \underset{y \in \Re^{n_2}}{\text{Minimize}} \quad g(y) && (4)\\
& \text{Subject to} \quad By = r - Cx && (5)\\
& \qquad\qquad\quad 0 \leq \mathrm{y} \leq v && (6)
\end{aligned}$$

If the second minimization problem is infeasible, we understand its value to be ∞.

In this formulation, let n_1 and n_2 denote the number of first-stage and second-stage decision variables, respectively, and let m_1 be the number of first-stage constraints (2), and m_2 be the number of second-stage constraints (5). The first-stage decision variables are $x \in \Re^{n_1}$, and the objective function component of these variables is $f : \Re^{n_1} \mapsto \Re$. The $m_1 \times n_1$ real matrix A and $b \in \Re^{m_1}$ specify constraints on the first-stage decision, and the vector $u^x \in \Re^{n_1}$ represents upper bounds on the first-stage variables.

The vector $y \in \Re^{n_2}$ represents the second-stage decisions. The $m_2 \times n_1$ real matrix C communicates information about the impact of the first-stage decision to the second-stage problem. The uncertainties of the second-stage scenario are represented by a (possibly) stochastic objective function $\mathbf{g} : \Re^{n_2} \rightarrow \Re$, the (possibly) stochastic $m_2 \times n_2$ real constraint matrix $\mathbf{B}$, and the (possibly) stochastic resource vector, $\mathbf{r} \in \Re^{m_2}$, and the vector $\mathbf{v} \in \Re^{n_1}$ of (possibly) stochastic upper bounds on the second-stage variables. Different instances of the stochastic objective function $\mathbf{g}$ may have different forms, e.g., quadratic, entropy or utility.

In this paper, we consider the case where the stochastic quantities $\mathbf{g}$, $\mathbf{r}$, $\mathbf{B}$ and $\mathbf{v}$ have a discrete and finite joint distribution, represented by the *scenario set* $< S >$. When this is the case, we can write

$$\mathcal{Q}(x) = \sum_{s=1}^{S} p^s Q(g^s, r^s, B^s, v^s \mid x), \tag{7}$$

where the probability of the realization of scenario s is

$$p^s = P\{(\mathbf{g}, \mathbf{r}, \mathbf{B}, \mathbf{v}) = (g^s, r^s, B^s, v^s)\}, \text{ for all } s \in < S >. \tag{8}$$

It is assumed that $p^s > 0$ for all $s \in < S >$ and that $\sum_{s=1}^{S} p^s = 1$.

Under the assumption of a finite, discrete event space, it is well known (Wets [1974]) that the stochastic nonlinear program [SNLP] can be reformulated to the equivalent large-scale deterministic nonlinear program:

[DNLP]

$$\underset{x \in \Re^{n_1}, y^s \in \Re^{n_2}}{\text{Minimize}} \quad f(x) + \sum_{s=1}^{S} p^s g^s(y^s) \tag{9}$$

$$\text{Subject to} \quad Ax = b \tag{10}$$

$$B^s y^s + c^s x = r^s \quad \text{for all } s \in < S > \tag{11}$$

$$0 \leq x \leq u^x \tag{12}$$

$$0 \leq y^s \leq v^s \quad \text{for all } s \in < S > \tag{13}$$

1.1.2 The Case of Network Problems

We now address the case where the problem constraint set ((2) and (5)) take the form of flow conservation constraints for some network problem The problem can be pure or generalized (i.e., it may have arc-multipliers). Specifically, we assume that the matrices

$$\begin{pmatrix} A \\ \hline C^s \end{pmatrix} \text{ and } B^s$$

are both node-arc incidence matrices for each scenario $s \in < S >$. Even with this assumption, the full problem [DNLP] is not a network problem due to the occurrence of $C^s x$ in (11). The first-stage variables x are, in this context, complicating variables and solution

approaches based on Benders' (or Generalized Benders') decomposition suggest themselves. An example of such an algorithm developed specifically for stochastic programs is the L-shaped decomposition method of Van Slyke and Wets [1969] and its regularized version, Ruszczynski [1986]. These methods, however, are based on cutting planes which destroy any structure present in the problem. Furthermore, it is not clear how one could parallelize these algorithms to any substantial degree. The progressive hedging algorithm of Rockafellar and Wets [1987] and the Augmented Lagrangian decomposition method of Ruszczynski [1989] seem to hold good promises for coarse-grain parallelization. First evidence for this observation was provided in Mulvey and Vladimirou [1989].

The algorithm we are developing transforms the original network problem [DNLP] into a large network with side-constraints which is also decomposable. This is achieved by *replicating* the first-stage variables x into a set of variables $x^s \in \Re^{n_1}$, for $s \in <S>$. Doing this — and adding the requirement that $x^1 = x^2 = ... = x^S$ — we obtain the equivalent nonlinear program

$$\textbf{[RNLP]} \quad \underset{x^s \in \Re^{n_1}, y^s \in \Re^{n_2}}{\text{Minimize}} \quad \sum_{s=1}^{S} p^s (f(x^s) + g^s(y^s)) \qquad (14)$$

$$\begin{aligned}
\text{Subject to} \quad & Ax^s = b \quad \text{for all } s \in <S> & (15)\\
& B^s y^s + c^s x^s = r^s \quad \text{for all } s \in <S> & (16)\\
& 0 \le x^s \le u^x \quad \text{for all } s \in <S> & (17)\\
& 0 \le y^s \le v^s \quad \text{for all } s \in <S> & (18)\\
& x^1 = x^s \quad \text{for all } s \in \{2, ..., S\} & (19)
\end{aligned}$$

By this reformulation, we have obtained a *network with side-constraints*. In the absense of the side-constraints (19), the problem decomposes completely into S nonlinear network problems. The algorithm we develop in the sequel capitalizes on this special structure by decomposing the problem into S network problems, iteratively solving these and then enforcing the side-constraints.

1.1.3 Matrix Structure

We show in Figure 1 the block matrix structure of [DNLP], excluding the simple bound constraints. Figure 2 similarly shows the structure of [RNLP]. It is evident from this figure that the problem decomposes by scenario if the non-anticipativity constraints are ignored.

Let $M = S \cdot (m^1 + m^2) + (S-1) \cdot m^1$, $N = S \cdot (n^1 + n^2)$ and let I denote the $n_1 \times n_1$ identity matrix. The block matrix for [RNLP] then has dimension $M \times N$. We denote this

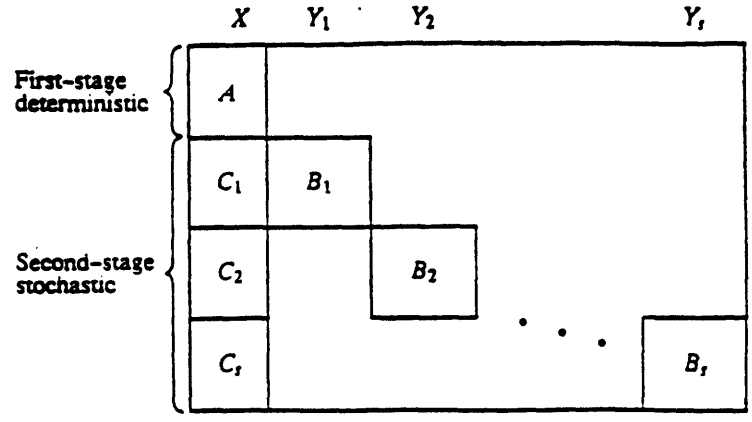


Figure 1: Block matrix structure of the deterministic equivalent nonlinear program [DNLP].

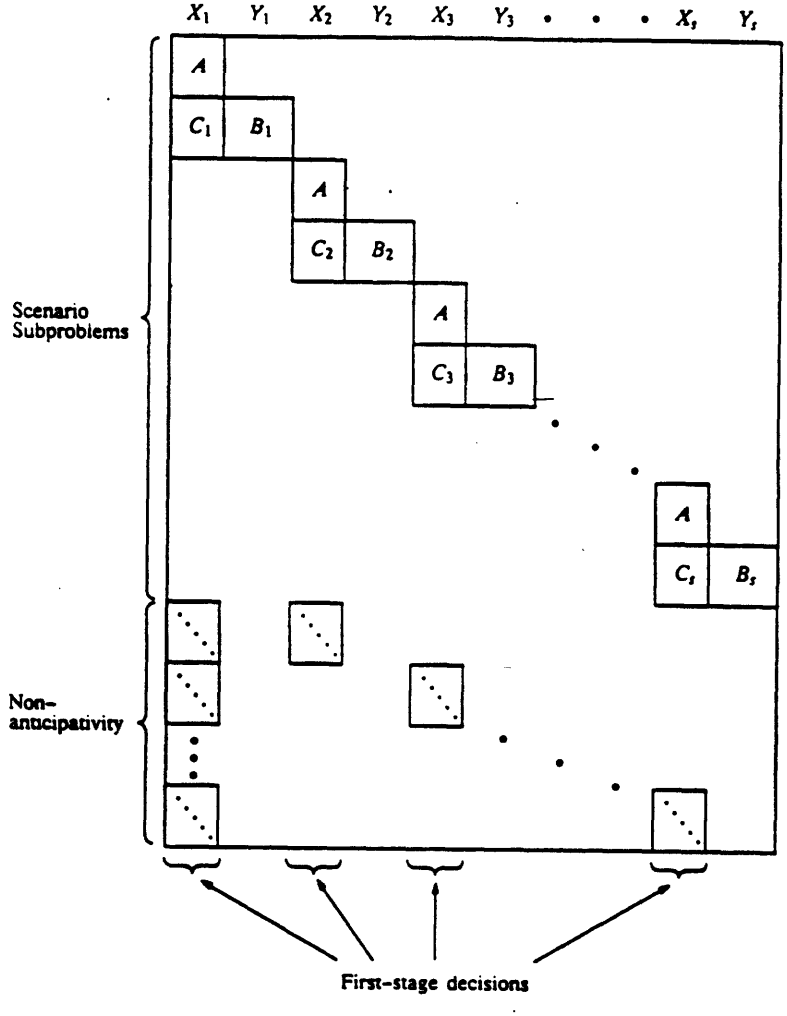


Figure 2: Block matrix structure of the replicated nonlinear program [RNLP].

matrix by Φ, that is,

$$\Phi = \begin{pmatrix} \begin{array}{|cc|} \hline A & \\ C^1 & B^1 \\ \hline \end{array} & & & & \\ & \begin{array}{|cc|} \hline A & \\ C^2 & B^2 \\ \hline \end{array} & & & \\ & & \ddots & & \\ & & & \begin{array}{|cc|} \hline A & \\ C^S & B^S \\ \hline \end{array} & \\ I & -I & & & \\ \vdots & & \ddots & & \\ I & & & -I & \end{pmatrix}. \tag{20}$$

We also denote by $\gamma \in \Re^M$ the right-hand side of [RNLP], i.e.,

$$\gamma = \begin{pmatrix} b \\ r^1 \\ \vdots \\ b \\ r^S \\ \hline 0 \\ \vdots \\ 0 \end{pmatrix}. \tag{21}$$

Similarly, we denote by $z \in \Re^N$ the set of decision variables,

$$z = \begin{pmatrix} x^1 \\ y^1 \\ \\ \vdots \\ x^S \\ y^S \end{pmatrix}, \tag{22}$$

and by $u \in \Re^N$ the upper bounds on z:

$$u = \begin{pmatrix} u^x \\ v^1 \\ \\ \vdots \\ u^x \\ v^S \\ \\ \end{pmatrix}. \tag{23}$$

Finally, we let $F(z)$ denote the objective function of [RNLP]:

$$F(z) = F(x^1, y^1, \cdots, x^S, y^S) = \sum_{s=1}^{S} p^s(f(x^s) + g^s(y^s)) \tag{24}$$

The replicated nonlinear program (14) - (19) can be written in compact form as

$$\textbf{[RNLP]} \quad \underset{z \in \Re^{n_1+n_2}}{\text{Minimize}} \quad F(z) \tag{25}$$

$$\text{Subject to} \quad \Phi z = \gamma \tag{26}$$

$$0 \leq z \leq u \tag{27}$$

We will be using the compact matrix notation in developing the row-action algorithm. However, the precise iterative steps of the algorithm depend on the network substructures of the matrix Φ. Hence, the algebraic formulation of the network substructures is given next.

1.1.4 Algebraic Representation of Network Problem

We assume for the sake of symmetry that the underlying network structure of all the scenario problems is the same. We denote this by the graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$, where $\mathcal{N} = <m_1 + m_2>$ is the set of nodes, and $\mathcal{E} \subseteq \mathcal{N} \times \mathcal{N}$ is the set of arcs. Let $\delta_i^+ = \{j \mid (i,j) \in \mathcal{E}\}$ be the set of nodes having an arc coming from node i, and $\delta_j^- = \{i \mid (i,j) \in \mathcal{E}\}$ be the set of nodes having an arc going to node i. We partition the node set into two disjoint sets, $\mathcal{N}^1$ and $\mathcal{N}^2$. $\mathcal{N}^1$ consists only of nodes incident to first-stage arcs, whereas nodes in $\mathcal{N}^2$ may have incident first- or second-stage arcs. We also partition the arc set $\mathcal{E}$ into two disjoint sets $\mathcal{E}^1$ and $\mathcal{E}^2$, corresponding to (replicated) first-stage and second-stage decisions, respectively. Denote by x_{ij}^s, $(i,j) \in \mathcal{E}^1$, and y_{ij}^s, $(i,j) \in \mathcal{E}^2$, the flow on the arc leading from node i to node j in scenario $s \in <S>$. (As in Section 1.1.1 we use x for first-stage and y for second-stage decisions). The upper bound of a (replicated) first-stage arc x_{ij}^s or a second-stage arc y_{ij}^s is denoted by u_{ij}^x and v_{ij}^s respectively. The multiplier on arc (i,j) is denoted by m_{ij}^s.

The network formulation corresponding to a single scenario $s \in <S>$ is now given by:

$$[\mathbf{NLP(s)}] \quad \text{Minimize} \sum_{\substack{x^s \in \Re^{n_1}, y^s \in \Re^{n_2} \\ (i,j) \in \mathcal{E}^1}} p^s f_{ij}(x_{ij}^s) + \sum_{(i,j) \in \mathcal{E}^2} p^s g_{ij}^s(y_{ij}^s) \tag{28}$$

$$\text{Subject to} \sum_{j \in \delta_i^+} x_{ij}^s - \sum_{k \in \delta_i^-} m_{ij}^s x_{ij}^s = b_i \quad \text{for all } i \in \mathcal{N}^1 \tag{29}$$

$$\sum_{j \in \delta_i^+ \cap \mathcal{N}^1} y_{ij}^s - \sum_{k \in \delta_i^- \cap \mathcal{N}^1} m_{ij}^s x_{ij}^s + \sum_{j \in \delta_i^+ \cap \mathcal{N}^2} y_{ij}^s - \sum_{k \in \delta_i^- \cap \mathcal{N}^2} m_{ij}^s x_{ij}^s = r_i \quad \text{for all } i \in \mathcal{N}^2 \tag{30}$$

$$0 \leq x_{ij}^s \leq u_{ij}^x \qquad \text{for all } (i,j) \in \mathcal{E}^1 \tag{31}$$

$$0 \leq y_{ij}^s \leq v_{ij}^s \qquad \text{for all } (i,j) \in \mathcal{E}^2 \tag{32}$$

The complete stochastic network problem [**RNLP**] is obtained by replicating this problem for each scenario $s \in <S>$ and including the non-anticipativity side constraints:

$$x_{ij}^1 = x_{ij}^s \quad \text{for all } s \in \{2, ..., S\} \text{ and } (i,j) \in \mathcal{E}^1. \tag{33}$$

We have in this section been referring to quantities pertaining to an arc $(i,j) \in \mathcal{E}$ under scenario $s \in <S>$ by using subscripts "(i,j)" and superscipt "s". We need to establish the correspondence between the vector notation of Section 1.1.3 and the algebraic notation of this section. If (i_1, j_1) denotes the first arc in $\mathcal{E}$ (see (22)), "z_1" and "$x_{i_1 j_1}^1$" refer to the same quantity. The connection is made formal in an obvious way by defining a numbering of the arcs in $\mathcal{E}$, e.g., the lexicographical ordering. Similar remarks apply for node-related (rather than arc-related) quantities.

2 The Row-Action Algorithm

The idea of the row-action algorithm is, as the name implies, to operate on one constraint of the problem at a time, simultaneously updating the primal variables occurring in the row, and the dual price of the row. Dual feasibility is maintained throughout the algorithm. Upon execution of a step, primal feasibility is obtained at the particular constraint. The algorithm terminates when primal feasibility is obtained for all constraints. The order in which the rows are operated on, the *control sequence*, is not formally important as long as no row is ignored indefinitely (although the ordering may influence the algorithmic performance). Such a control sequence is called *almost cyclic*, Censor and Lent [1981] and Censor [1981].

In this section we present the general algorithm and its specialization to stochastic network problems of the form [RNLP]. We also demonstrate the potential for parallellism, even massive parallellism, that results from the application of the algorithm.

2.1 The General Row-Action Framework

We now present, in a general form, the row-action algorithm. Let $F : \Lambda \subseteq \Re^n \mapsto \Re$ and let $S \neq \emptyset$ be an open convex set such that $\bar{S} \in \Lambda$. 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 $D(x,y) = F(x) - F(y) - \nabla F(y)^T(x-y)$, and let $H(a,b)$ be the hyperplane $H(a,b) = \{x \in \Re^n \mid a^T x = b\}$. The D-projection (or Bregman projection) of a point y onto $H(a,b)$ is defined by

$$P_H(a,b) = \arg \min_{x \in H(a,b) \cap \bar{S}} D(x,y). \tag{34}$$

A function F which 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

$$\nabla F(x) = \nabla F(y) + \beta \cdot a \tag{35}$$

$$a^T x = b. \tag{36}$$

The real number β is known as the Bregman parameter.

We are now in a position to present the row-action algorithm for the constraints specified by the rows of Φ and the corresponding elements of γ (26), and by the bound constraints on z, (27).

After initialization, the algorithm proceeds by projecting upon one constraint at a time, updating the dual price of the constraint and the primal variables occurring in the constraint in order to maintain dual feasibility. We express the order in which constraints are considered using a *control sequence*, $\ell(\nu)$, such that constraint $\ell(\nu)$ is considered at iteration ν. If $\ell(\nu) \in \{1, ..., M\}$ the constraint considered is $H(\Phi^{l(\nu)}, \gamma_{l(\nu)})$. If $\ell(\nu) \in \{M+1, ..., M+N\}$ we consider the interval constraint on the $(\ell(\nu) - M)^{\text{th}}$ variable, see (27). For clarity of notation we will abbreviate $\ell(\nu)$ by ℓ.

Let Φ^ℓ denote the ℓ^{th} row of Φ and let γ_ℓ denote the ℓ^{th} element of γ. Assuming that $F(z)$ is a Bregman's function and has the strong zone consistency property with respect to the hyperplanes $H(\Phi^\ell, \gamma_\ell)$, we can state the general algorithm as follows:

General Row-Action Algorithm

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

$$\nabla F(z^0) = -\Phi^T \pi^0. \tag{37}$$

Step 1: (Iterative step over equality constraints). For $\ell \in \{1, 2, ..., M\}$ solve for $z^{\nu+1}$ and β^ν the equations

$$\begin{aligned} \nabla F(z^{\nu+1}) &= \nabla F(z^\nu) + \beta^\nu \Phi^\ell, & (38) \\ z^{\nu+1} &\in H(\Phi^\ell, \gamma_\ell). & (39) \end{aligned}$$

Update the dual price:

$$\pi^{\nu+1} = \pi^\nu - \beta^\nu e^\ell, \tag{40}$$

Step 2: (Iterative step over simple bound constraints). For $\ell \in \{M+1, ..., M+N\}$ project $z^\nu_{\ell-M}$ upon its bounds:

If $z^\nu_{\ell-M} \leq 0$, let β^ν and $z^{\nu+1}$ be the solution of

$$\begin{aligned} \nabla F(z^{\nu+1}) &= \nabla F(z^\nu) + \beta^\nu \Phi^\ell, & (41) \\ z^{\nu+1}_{\ell-M} &= 0 & (42) \end{aligned}$$

If $z^\nu_{\ell-M} \geq u_{\ell-M}$, let β^ν and $z^{\nu+1}$ be the solution of

$$\begin{aligned} \nabla F(z^{\nu+1}) &= \nabla F(z^\nu) + \beta^\nu \Phi^\ell, & (43) \\ z^{\nu+1}_{\ell-M} &= u_{\ell-M} & (44) \end{aligned}$$

If $0 < z^\nu_{\ell-M} < u_{\ell-M}$, let β^ν and $z^{\nu+1}$ be the solution of

$$\begin{aligned} \nabla F(z^{\nu+1}) &= \nabla F(z^\nu) + \beta^\nu \Phi^\ell, & (45) \\ \beta^\nu &= \pi^\nu_\ell. & (46) \end{aligned}$$

Update the dual price:

$$\pi^{\nu+1} = \pi^\nu - \beta^\nu e^\ell, \tag{47}$$

Step 3 Set $\nu \leftarrow \nu + 1$ and proceed from Step 1.

2.2 Specialization to Quadratic Stochastic Networks

In this section we specialize the row-action algorithm to the case of quadratic network flow problems with the non-anticipativity constraints. We do not distinguish in this section between first-stage and second-stage variables, but use "x_{ij}^s" for both sets of variables. We thus assume that F takes the form

$$F(x) = \sum_{(i,j)\in\mathcal{E}, s\in<S>} p^s(\frac{1}{2}w_{ij}^s(x_{ij}^s)^2 + q_{ij}^s x_{ij}^s). \tag{48}$$

for $w_{ij}^s > 0$. Let $M^1 = S \cdot (m^1 + m^2)$. Then rows $1, ..., M^1$ of the constraint matrix Φ are network flow conservation constraints, and rows $M^1 + 1, ..., M$ for the nonanticipativity constraints take the simple form

$$x_{i_1 j_1}^s - x_{i_2 j_2}^s = 0$$

for some $(i_1, j_1), (i_2, j_2) \in \mathcal{E}$ and $s \in <S>$. We now proceed to develop the specific projection formulae for use in Steps 1 and 2 of the general row-action algorithm. The complete algorithm is summarized in Section 2.3.

2.2.1 Projection on Flow Conservation Constraints

First we derive the projection upon flow conservation constraints of a generalized network ((29) and (30)). We consider in this section a single node, $i \in \mathcal{N}$, the incoming arcs, x_{ij} for $j \in \delta_i^-$, and the outgoing arcs, x_{ij} for $j \in \delta_i^+$ under some scenario $s \in <S>$.

The Bregman projection x^s of the current iterate y^s upon the hyperplane $H(\Phi^i, \gamma_i)$ determined by the flow conservation constraint on node i is the solution to

$$\nabla F(x^s) = \nabla F(y^s) + \beta \cdot \Phi^i, \tag{49}$$

$$x^s \in H(\Phi^i, \gamma_i). \tag{50}$$

Of course, if $y^s \in H(\Phi^i, \gamma_i)$, then $\beta = 0$. If the current iterate does not satisfy flow conservation, we define the *node surplus* s_i as

$$s_i = b_i - (\sum_{j\in\delta_i^+} y_{ij}^s - \sum_{k\in\delta_i^-} m_{ki}^s y_{ki}^s). \tag{51}$$

Writing out (49) in full, we obtain

$$x_{ij}^s = y_{ij}^s + \beta \cdot \frac{1}{w_{ij}^s} \text{ for } j \in \delta_i^+, \tag{52}$$

$$x_{ki}^s = y_{ki}^s - \beta \cdot \frac{m_{ki}^s}{w_{ki}^s} \text{ for } k \in \delta_i^-. \tag{53}$$

This solution must satisfy (50), i.e.,

$$\sum_{j\in\delta_i^+} (x_{ij}^s + \beta \cdot \frac{1}{w_{ij}^s}) - \sum_{k\in\delta_i^-} m_{ki}(x_{ki}^s - \beta \cdot \frac{m_{ki}^s}{w_{ki}^s}) = b_i. \tag{54}$$

From this and (51) we get

$$\beta = \frac{s_i}{\sum_{j\in\delta_i^+} \frac{1}{w_{ij}^s} + \sum_{k\in\delta_i^-} \frac{(m_{ki}^s)^2}{w_{ki}^s}} \tag{55}$$

Using this result in (52) and (53) gives us the desired formula for updating all primal variables incident to node i. The dual variable for this node is updated by subtracting β from its current value, $\pi_i^s \leftarrow \pi_i^s - \beta$.

2.2.2 Projection on Simple Bound Constraints

We now develop the specific projections on the simple bounds, (31) and (32) corresponding to Step 2 of the general row-action algorithm. We consider a variable x_{ij}^s and the upper bound u_{ij}^x.

Denote by y_{ij}^s the value of the variable at the previous iteration, and by x_{ij}^s the projected value. If $y_{ij} < 0$, we get from (41) and (42):

$$0 = x_{ij}^s = y_{ij}^s + \frac{\beta}{w_{ij}^s}\Phi' \tag{56}$$

The primal variable is thus set to 0. Solving, we get the Bregman parameter to be

$$\beta = -w_{ij}^s y_{ij}^s. \tag{57}$$

The dual price of the constraint is updated by subtracting β from its current value, $\pi_{ij}^s \leftarrow \pi_{ij}^s - \beta$.

If $y_{ij}^s > u_{ij}^x$ we similarly set the primal variable x_{ij}^s to the upper bound, u_{ij}^x, and find the Bregman parameter to be

$$\beta = (x_{ij}^s - u_{ij}^x)w_{ij}^s \tag{58}$$

and update the dual price of the bound constraint by subtracting β from it.

Finally, if $0 \le y_{ij}^s \le u_{ij}^x$ we get

$$x_{ij}^s = y_{ij}^s + \frac{\pi_{ij}^s}{w_{ij}^s} \tag{59}$$

and then set the dual price π_{ij}^s to 0.

2.2.3 Projections on Nonanticipativity Constraints

A non-anticipativity constraint (33) takes the form

$$x_{ij}^1 - x_{ij}^s = 0 \tag{60}$$

for some $(i,j) \in \mathcal{E}$ and some $s \in \{2,...,S\}$. If y is the current iterate, the Bregman projection upon this constraint solves

$$\nabla F(x) = \nabla F(y) + \beta d^s, \tag{61}$$

$$x^1_{ij} = x^s_{ij}, \tag{62}$$

where d^s is the vector having a "1" in position 1 and a "-1" in position s, and x is the projected point. This system can be written as

$$x^1_{ij} = x^s_{ij} = y^1_{ij} + \frac{\beta}{p^s w^1_{ij}} = y^s_{ij} - \frac{\beta}{p^s w^s_{ij}}. \tag{63}$$

Solving this, we get

$$x^1_{ij} = x^s_{ij} = \frac{p^s w^1_{ij} y^1_{ij} + p^s w^s_{ij} y^s_{ij}}{p^s w^1_{ij} + p^s w^s_{ij}}, \tag{64}$$

i.e., the point (y^1_{ij}, y^s_{ij}) is projected upon the point with coordinates equal to the weighted average of y^1_{ij} and y^s_{ij}, where $p^s w^1_{ij}$ and $p^s w^s_{ij}$ are the weights. In the next section, we will provide a significant generalization of this result to the case of all S first-stage variables rather than just a pair of them.

2.2.4 Closed Form Solution for Non-anticipativity Constraints

In this section we obtain a closed form solution for the projection on all S non-anticipativity constraints. This result has important implications for the massively parallel implementation of the algorithm. We consider the effect of repeated projections on a subset of the non-anticipativity constraints (33). In particular, consider the subset which, for a given arc (i,j), enforces equality of the scenario replications, i.e., $x^s_{ij} = x^{s'}_{ij}$ for all $s, s' \in <S>$. The almost cyclic control framework of the row-action algorithm allows repeated projections upon only these constraints until convergence (within some tolerance) of the variables x^s_{ij} to a common value, $\hat{x}_{ij}$. We show that $\hat{x}_{ij}$ can be obtained analytically rather than using the iterative scheme.

The non-anticipativity constraints for a first-stage variable x_{ij} take the form

$$\begin{aligned} x^1_{ij} &= x^2_{ij}, \\ x^1_{ij} &= x^3_{ij}, \\ &\cdots \\ x^1_{ij} &= x^S_{ij}. \end{aligned} \tag{65}$$

By repeated projection upon these constraints, such that the ν^{th} projection is upon the hyperplane $H(\Phi^\ell(\nu), \gamma_\ell(\nu))$ we obtain a sequence of points $x^\nu \in \Re^S$ satisfying

$$\nabla F(x^\nu) = \nabla F(y) + \sum_{k=1}^{\nu} \lambda^k e^{\ell(k)} \tag{66}$$

where λ_ν is the Bregman parameter corresponding to the ν^{th} projection, and y is the starting point. The limiting point x^* satisfies

$$\nabla F(x^*) = \nabla F(y) + \sum_{k=1}^{\infty} \lambda^k e^{\ell(k)} \tag{67}$$

and must, by (65), have all components identical, i.e., $x^* = (\hat{x}_{ij}, ..., \hat{x}_{ij})$.

Let

$$\Lambda^s = \sum_{\{k|\ell(k)=s\}} \lambda^k$$

for $k = 2, ..., S$. Using the fact that $F(y)$ is a quadratic function, as given in (48), rewrite (67) as the square system in S variables, $\hat{x}_{ij}, \Lambda^2, ..., \Lambda^S$:

$$\begin{aligned}
\hat{x}_{ij} &= y^1_{ij} + \frac{1}{p^1 w^1_{ij}} \sum_{s=2}^{S} \Lambda^s, \\
\hat{x}_{ij} &= y^2_{ij} - \frac{1}{p^2 w^2_{ij}} \Lambda^2, \\
&\cdots \\
\hat{x}_{ij} &= y^S_{ij} - \frac{1}{p^S w^S_{ij}} \Lambda^S.
\end{aligned}$$

In matrix form, this is

$$Ht = y \tag{68}$$

where

$$H = \begin{pmatrix}
1 & \frac{-1}{p^1 w^1_{ij}} & \frac{-1}{p^1 w^1_{ij}} & \cdots & \frac{-1}{p^1 w^1_{ij}} \\
1 & \frac{1}{p^2 w^2_{ij}} & & & \\
1 & & \frac{1}{p^3 w^3_{ij}} & & \\
\cdots & & & \ddots & \\
1 & & & & \frac{1}{p^S w^S_{ij}}
\end{pmatrix}, \tag{69}$$

and $t = (\hat{x}_{ij}, \Lambda^2, ..., \Lambda^S)^T$. By inverting H we can solve for t. Since we are only interested in $\hat{x}_{ij}$, not $\Lambda^2, ..., \Lambda^S$, we need only calculate the first row of H^{-1}, denoted by h. Due to the special structure of H, we easily get

$$h = \frac{1}{\det(H)} \cdot \prod_{s=1}^{S} \frac{1}{p_s w^s_{ij}} \cdot (p^1 w^1_{ij}, p^2 w^2_{ij}, ..., p^S w^S_{ij})^T,$$

where $\det(H)$ is the determinant of H. The inner product of the first column of H, which consists of all ones, and the first row of H^{-1} must equal 1, and therefore $\sum_{s=1}^{S} h^s = 1$. Hence

$$\det(H) = \prod_{s=1}^{S} \frac{1}{p^s w^s_{ij}} \cdot \sum_{s=1}^{S} p^s w^s_{ij}.$$

Note that $\det(H) > 0$, so system (68) has a unique solution. Solving for $\hat{x}_{ij}$ we get

$$\hat{x}_{ij} = h^T y = \frac{\sum_{s=1}^{S} p^s w_{ij}^s y_{ij}^s}{\sum_{s=1}^{S} p^s w_{ij}^s}. \tag{70}$$

Since x is a first-stage variable, $w_{ij}^1 = w_{ij}^2 = ... = w_{ij}^S$. Also, $\sum_{s=1}^{S} p^s = 1$, so the result can be simplified to

$$\hat{x}_{ij} = \sum_{s=1}^{S} p^s y_{ij}^s \tag{71}$$

2.3 The Row-Action Algorithm for Quadratic Stochastic Networks

We have now completed all the components required for the row-action algorithm applied to the quadratic stochastic network. The complete algorithm proceeds as follows:

Row-Action Algorithm for Quadratic Stochastic Networks

Step 0: (Initialization) $\nu - 0$. Get π^0 and z^0 such that $\nabla F(z^0) = -\Phi^T \pi^0$. For example, $\pi^0 = 0$ and

$$(x_{ij}^s)^0 = -\frac{q_{ij}}{w_{ij}} \text{ for all } (i,j) \in \mathcal{E}^1, s \in <S>, \tag{72}$$

$$(y_{ij}^s)^0 = -\frac{q_{ij}^s}{w_{ij}^s} \text{ for all } (i,j) \in \mathcal{E}^2, s \in <S>. \tag{73}$$

Step 1: (Solve scenario subproblems). For all $s \in <S>$:

Step 1.1: (Solve for flow conservation constraints) Let

$$(\beta_i^s)^\nu = \frac{s_i}{\sum_{j \in \delta_i^+} \frac{1}{w_{ij}^s} + \sum_{k \in \delta_i^-} \frac{(m_{ki}^s)^2}{w_{ki}^s}} \text{ for all } i \in \mathcal{N}^1 \cup \mathcal{N}^2. \tag{74}$$

For all first-stage nodes $i \in \mathcal{N}^1$:

$$(x_{ij}^s)^{\nu+1} = (x_{ij}^s)^\nu + (\beta_i^s)^\nu \cdot \frac{1}{w_{ij}^s} \text{ for all } j \in \delta_i^+, \tag{75}$$

$$(x_{ki}^s)^{\nu+1} = (x_{ki}^s)^\nu - (\beta_i^s)^\nu \cdot \frac{m_{ki}^s}{w_{ki}^s} \text{ for all } k \in \delta_i^- \tag{76}$$

$$\pi_i^{\nu+1} = \pi_i^\nu - (\beta_i^s)^\nu. \tag{77}$$

For all second-stage nodes $i \in \mathcal{N}^2$:

$$(y_{ij}^s)^{\nu+1} = (y_{ij}^s)^\nu + (\beta_i^s)^\nu \cdot \frac{1}{w_{ij}^s} \text{ for all } j \in \delta_i^+, \tag{78}$$

$$(y_{ki}^s)^{\nu+1} = (y_{ki}^s)^\nu - (\beta_i^s)^\nu \cdot \frac{m_{ki}^s}{w_{ki}^s} \text{ for all } k \in \delta_i^- \tag{79}$$

$$\pi_i^{\nu+1} = \pi_i^\nu - (\beta_i^s)^\nu. \tag{80}$$

Step 1.2: (Solve for the simple bounds).

For all first-stage variables $(i,j) \in \mathcal{E}^1$:

$$(x_{ij}^s)^{\nu+1} = \begin{cases} u_{ij}^x & \text{if } (x_{ij}^s)^\nu \geq u_{ij}^x, \\ 0 & \text{if } (x_{ij}^s)^\nu \leq 0, \\ \pi_{ij}^\nu & \text{if } 0 < (x_{ij}^s)^\nu < u_{ij}^x. \end{cases} \tag{81}$$

and

$$\pi_{ij}^{\nu+1} = \begin{cases} \pi_{ij}^\nu - w_{ij}(u_{ij}^x - (x_{ij}^s)^\nu) & \text{if } (x_{ij}^s)^\nu \geq u_{ij}^x, \\ \pi_{ij}^\nu + w_{ij}^s(x_{ij}^s)^\nu & \text{if } (x_{ij}^s)^\nu \leq 0, \\ 0 & \text{if } 0 < (x_{ij}^s)^\nu < u_{ij}^x. \end{cases} \tag{82}$$

For all second-stage arcs $(i,j) \in \mathcal{E}^2$:

$$(y_{ij}^s)^{\nu+1} = \begin{cases} v_{ij} & \text{if } (y_{ij}^s)^\nu \geq v_{ij}, \\ 0 & \text{if } (y_{ij}^s)^\nu \leq 0, \\ \pi_{ij}^\nu & \text{if } 0 < (y_{ij}^s)^\nu < v_{ij}. \end{cases} \tag{83}$$

and

$$\pi_{ij}^{\nu+1} = \begin{cases} \pi_{ij}^\nu - w_{ij}^s(v_{ij}^s - (y_{ij}^s)^\nu) & \text{if } (y_{ij}^s)^\nu \geq v_{ij}, \\ \pi_{ij}^\nu - w_{ij}^s(y_{ij}^s)^\nu & \text{if } (y_{ij}^s)^\nu \leq 0, \\ 0 & \text{if } 0 < (y_{ij}^s)^\nu < v_{ij}. \end{cases} \tag{84}$$

Step 2: (Solve for non-anticipativity constraints):

For all first-stage arcs $(i,j) \in \mathcal{E}^1$:

$$\hat{x}_{ij} = \sum_{s=1}^{S} p^s (x_{ij}^s)^\nu, \tag{85}$$

$$(x_{ij}^s)^{\nu+1} = \hat{x}_{ij} \text{ for all } s \in <S>. \tag{86}$$

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

2.4 Potential for Parallellism

The general row-action algorithm as specialized to the case of stochastic network problems exhibits potential for parallel execution at several levels.

First, it is clear from Figure 2 that by ignoring the non-anticipativity constraints the resulting problem decomposes by scenario. Assigning a processor to execute the row-action iterates for each scenario allows for *coarse-grained* parallel execution. Projection on the non-anticipativity constraints requires accumulation of the value of replicated variables from all scenarios. This accumulation can be implemented very efficiently on most distributed architectures (e.g., hypercubes) and is of course trivial on shared memory architectures. Second, the row-action projection can obviously be executed concurrently on rows which do not have common variables, allowing for *fine-grained* parallellism. For the network parts of the algorithm (i.e., for each scenario subproblem), this corresponds to concurrent execution for sets of nodes which are disconnected. The problem of finding such sets of nodes for the purpose of implementing parallel algorithms was recognized by Zenios and Mulvey [1988] to be equivalent to the graph-coloring problem (Bertsekas and Tsitsiklis [1989]), and can be solved very efficiently using a heuristic.

The opportunities for parallellism noted above do not destroy the convergence properties of the row-action algorithm, since they correspond to a Gauss-Seidel type execution. By relaxing the requirement that parallel execution be performed only on rows which do not have common variables, one can obtain a Jacobi-type algorithm (Bertsekas and Tsitsiklis [1989]), allowing for an even larger degree of parallellism. This idea forms the basis of massively parallel algorithms for solving network problems (Zenios and Lasken [1988], Zenios and Nielsen [1990]) and multicommodity network problems (Zenios [1990]) as well as for the network solvers used in this study.

3 Massively Parallel Implementation

Our primary motivation in developing the row-action algorithm for stochastic network problems was the desire to exploit massively parallel computing for solving very large problems. The algorithm naturally decomposes in a way making it suitable for solution on a SIMD-type (Single Instruction, Multiple Data) computer. This section discusses data-structures for the massively parallel implementation of the algorithm on a Connection Machine CM-2. First, we give a brief description of the CM-2.

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 32Kbytes, 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 applications. 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.

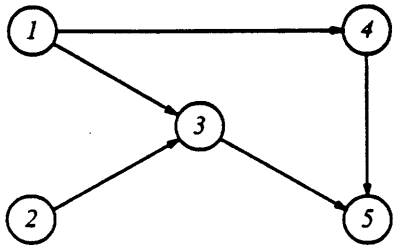
Before invoking Paris instructions from a program the user has to specify the VP set, create a geometry, and associate the VP set with the geometry. Thus a communications mechanism is established (along both *send* and NEWS addresses). Paris instructions — parallel primitives — can then be invoked to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

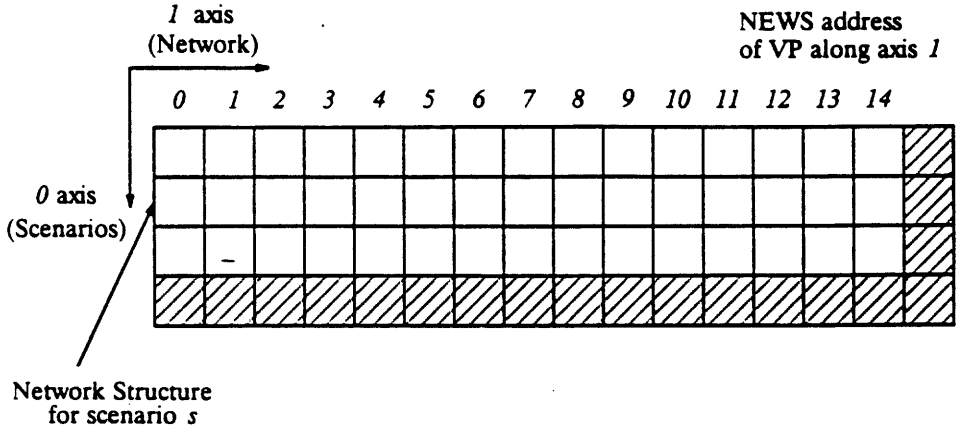
Scan is also known in the literature as parallel prefix. The $\otimes$-scan primitive, for an associative, binary operator $\otimes$, takes a sequence $\{x_0, x_1, \ldots, x_n\}$ and produces another sequence $\{y_0, y_1, \ldots, y_n\}$ such that $y_i = x_0 \otimes x_1 \otimes \cdots \otimes x_i$. On 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 Data-level Parallel Representation of Sparse Stochastic Networks

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 repre-





Data Fields in the s-th row corresponding to scenario s.

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/Demand	$s_k(1)$	$s_k(1)$	$s_k(1)$	$s_k(2)$	$s_k(2)$	$s_k(3)$	$s_k(3)$	$s_k(3)$	$s_k(3)$	$s_k(4)$	$s_k(4)$	$s_k(4)$	$s_k(5)$	$s_k(5)$	$s_k(5)$
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
Stage bit	0	1	0	0	1	0	1	1	0	0	0	0	0	0	0

Figure 3: Representing stochastic network problems on the CM.

sent sparse network problems. A comparison of alternative parallel implementations is reported in Nielsen and Zenios [1990]. These data structures have been employed 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 $\lceil 2(m_1 + m_2) + n_1 + n_2 \rceil_2$ ($\lceil \cdot \rceil_2$ denotes rounding up to the nearest integer power of 2). 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, stochastic network solver, we use the nonlinear network optimizer of Zenios and Nielsen [1990]. This solver is designed to handle sparse transhipment problems. Interestingly, this solver can be easily extended to solve multiple independent scenarios in parallel: The CM is configured as a two-dimensional NEWS geometry, of dimensions $\lceil S \rceil_2 \times \lceil 2(m_1+m_2)+n_1+n_2 \rceil_2$. Each row of the 0-axis is used to represent a single network problem as outlined above. Since the network problem has identical topology under all the scenarios, 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 s of the 0-axis will store the data of the network problem for the s-th scenario. This configuration is illustrated in Figure 3. The algorithm iterates along the 1-axis until some convergence criteria is satisfied for all the rows. Once the single scenario networks are solved by iterations along the 1-axis, the algorithm executes the projection on the non-anticipitativity constraints using scan operations along the 0-axis as explained in the next section.

In the implementation we created the geometry by assigning virtual processors to physical processors in such a way that communication along the axis of the NEWS grid holding individual networks (the dominant cummunication axis) was favored. Doing this rather than relying on the default VP-assignment speeds up the algorithm by a factor of approximately 1.8 for a VP-ratio of 16, and approximately 1.4 for a VP-ratio of 4.

Each projection on the non-anticipativity constraints is called a major iteration. Iterations for the solution of the scenario subproblems are called minor.

3.3 Projection on Non-anticipativity Constraints

The projection on the non-anticipativity constraints (85) is executed in parallel for all first-stage (replicated) variables. Each first-stage variable (with replications) occupies two columns of the two-dimensional NEWS grid representing the stochastic program (Figure 3). Each processor holds the scenario probability p^s and the component x^s_{ij} of the current iterate, and computes their product. The products are then added and distributed back to each processor (using `SPREAD-WITH-ADD`) as the projected point, $\hat{x}_{ij}$.

The network solver is essentially a "black box" which, given any dual starting point, will return a dual and primal feasible solution. In contrast, the row-action algorithm prescribes the use of projections, which are dependent on the starting point, on the network flow-conservation constraints. To achieve compatibility between the network solver and the row-action algorithm we perturb the objective function for the first-stage variables for each scenario subproblem. This is done after every projection on the non-anticipativity constraints, i.e., after every major iteration..

Let the objective function component for x_{ij}^s be

$$f_{ij}^s(x_{ij}^s) = p^s(\frac{1}{2}w_{ij}(x_{ij}^s)^2 + q_{ij}x_{ij}^s).$$

Based on the projection point $\hat{x}_i$ we perturb the objection function component to

$$\hat{f}_{ij}^s(x_{ij}^s) = p^s(\frac{1}{2}w_{ij}(x_{ij}^s)^2 + \hat{q}_{ij}x_{ij}^s),$$

where

$$\hat{q}_{ij} = q_{ij} + w_{ij}(x_{ij}^s - \hat{x}_i j). \tag{87}$$

The equivalence of the updating formula (87) followed by the "black box" network algorithm to the row-action type projections can be established by noting that the projected point $\hat{x}_{ij}$ must still satisfy complementary slackness, i.e., solve

$$p^s(w_{ij}\hat{x}_{ij}^s + \hat{q}_{ij}) = t_{ij}^s = p^s(w_{ij}x_{ij}^s + q_{ij}),$$

where $t^s = -\Phi^T\pi$ is the *tension* on the arcs of scenario s (see Bertsekas and Tsitsiklis [1989]).

4 Experiments and Numerical Results

The row-action algorithm for solving stochastic network problems was implemented on the Connection Machine CM-2 The program was written in C/Paris, and was run using a VAX 8800 front-end at Nort-east Parallel Architectures Center (NPAC) under microcode version 5211. In this section we report on the performance of the algorithm on a number of test problems from financial applications. We also investigate the effect of stochasticity in the network parameters on the performance of the algorithm.

4.1 Test Problems

We use as primary test problems a set of asset allocation models from Mulvey and Vladimirou [1989]. An investor distributes his wealth among a set of assets which have uncertain returns. His goal is to maximize the expected utility of final wealth after the end of the time horizon, which consists of a number of periods. He has the option to redistribute investments among assets between time periods. The investment decisions in

Problem	Assets	Horizon	Scenarios	Nodes	Arcs	Equivalent NLP size
Deter 0	15	8	18	121	334	2178 x 6012
Deter 1	15	6	52	121	335	6292 x 17420
Deter 2	15	8	80	91	249	7280 x 19920
Deter 3	15	8	72	121	335	8712 x 24120
Deter 4	15	4	70	61	163	4270 x 11410
Deter 5	15	8	48	121	335	5808 x 16080
Deter 6	15	8	40	121	335	4840 x 13400
Deter 7	15	8	60	121	335	7260 x 20100
Deter 8	15	8	36	121	335	4356 x 12060

Table 1: Characteristics of the test problems Deter 0 – Deter 8.

Problem	Major Its.	Minor Its.	Time	VP ratio
Deter 0	9	850	10.5	4
Deter 1	9	875	19.0	8
Deter 2	7	625	22.0	16
Deter 3	9	875	31.4	16
Deter 4	5	325	6.7	8
Deter 5	10	950	24.7	8
Deter 6	9	900	19.4	8
Deter 7	11	1025	22.3	8
Deter 8	11	1050	22.7	8

Table 2: Solution time of test problems on the CM-2 with 8K processing elements (in seconds).

CM–2 size	VP-ratio 4	VP-ratio 8	VP-ratio 16
8K PEs	45.1	50.4	62.7
64K PEs	360.7	403.5	501.5

Table 3: MFLOP rate for different VP ratios and CM–2 sizes

period 1 are the first-stage variables. The multipliers (asset returns) are stochastic, in the range 0.94 to 1.06. Due to the requirement that the objective must be quadratic, we minimize $\sum_{(i,j)\in\mathcal{E},s\in S}(x_{ij}^s)^2$. Table 1 list the characteristics of these test problems. The size of the deterministic equivalent nonlinear program is also shown.

The problems were solved on an 8K PE CM–2. The algorithm was terminated when both the absolute node surplus/deficit was below a small tolerance $\epsilon > 0$ for each node, and each non-anticipativity constraint was violated by less than ϵ, for $\epsilon = 10^{-3}$. Results are shown in Table 2. We imposed a limit of 100 minor iterations between performing a projection on the nonanticipativity constraints (constituting a major iteration). Convergence of the network subproblems (to within the tolerance) was checked every 25 iterations.

The results show that all of these problems were solved in less than a half minute, and required only up to 11 major iterations. We project that these problems could be solved in under 5 seconds each on a full 64K CM–2. It is also interesting to observe that the solution time does not depend on the number of scenarios, or the size of the equivalent nonlinear program. For example, Deter4 solves in much less time than Deter6 although it has twice as many scenarios.

For comparison, we solved the test problem deter0 using GAMS/Minos 5.1 on a VAX 6000. Minos terminated with an optimal solution after 8.8 hours of computer time and 29153 iterations. Due to the overhead of the GAMS interpreter, a stand-alone version of Minos 5.1 would solve this problem in about 4 hours. In comparison, Carpenter et al. [1990] solve this problem in 506 seconds using the OBN code on an IRIS4D/70 workstation.

4.2 MFLOP rate on the Connection Machine CM–2

Based on the solution of large-scale problems (Table 4) we can calculate the computational rate of the algorithm. Each VP performs 17 floating point operations per minor iteration. The work involved in the major iterations is ignored since it is very little, and only performed once every 100 minor iterations. For deter0, which ran at a VP-ratio of 4 on an 8K machine and performed 850 iterations in 10.5 seconds, we thus obtained

$$\frac{8192 \cdot 17 \cdot 4 \cdot 850}{10.5} = 45.1 \text{ MFLOPS.}$$

By scaling this result to a 64K CM–2, we obtain a computing rate of 360.7 MFLOPS.

The MFLOP rate is largely dependent on the network structure, which determines the time to execute the general router communication, and the VP-ratio: The CM–2 performs

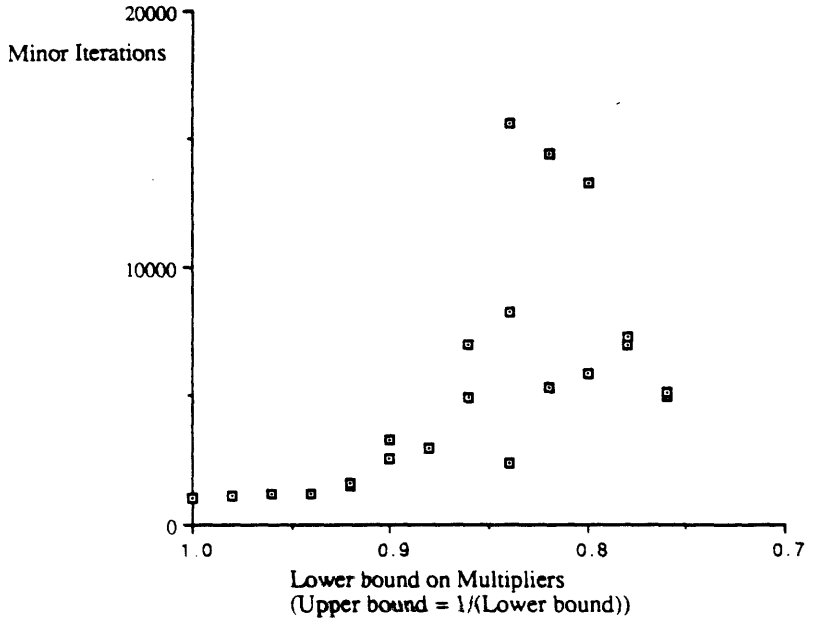


Figure 4: Effect on the algorithm of range of multipliers.

slightly better at higher VP-ratios. We summarize in Table 3 the average MFLOP rates obtained for the test problems deter0 - deter8. Due to the fact that the two-dimensional NEWS-grid must have coordinate dimensions which are powers of two, the actual useful MFLOP rate (i.e., contributing toward a problem solution) could be up to a factor four less than shown in the table.

4.3 Effects of Stochasticity in Multipliers

The deter0 - deter8 test problems all have multipliers in a fairly limited range, and all the objective coefficients were the same. To investigate the behavior of the algorithm when the multipliers vary more than in these problems, we generated a number of test problems with random multipliers.

Figure 4 shows the number of minor iterations required to solve variants of deter0 where the multipliers were all randomly generated in the interval [lb,1/lb], where lb is on the ordinate axis. The algorithm behaves well for multipliers within the range [0.85, 1.18], but then becomes unstable: For some problems, it still converges within a reasonable number of iterations, but for some problems, convergence becomes very slow. Generally, the number of iterations seems to increase exponentially with the range of the multipliers.

For some of the runs, the number of major iterations was quite high, whereas the network subproblems converged fast. For other problems the reverse was true. This indicates that the distribution of the random coefficients across the scenarios is significant: If each

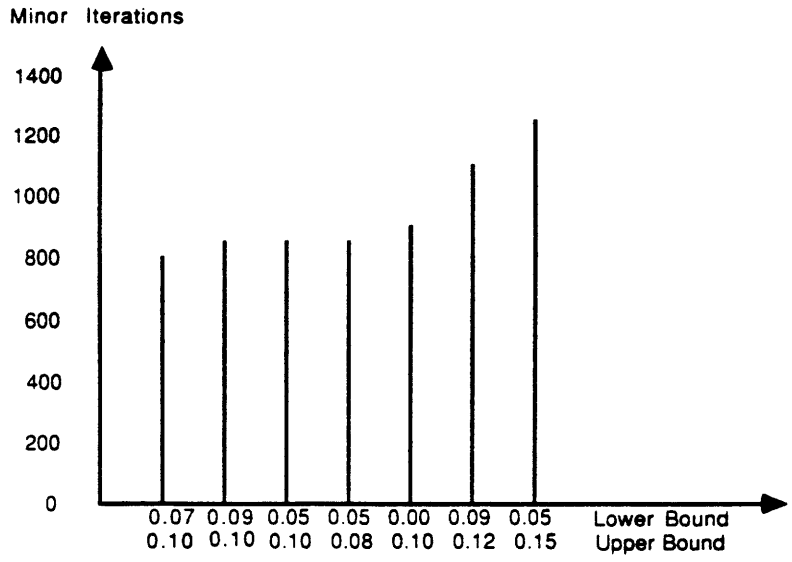


Figure 5: Effect on iterations of stochastic right-hand sides

scenario network is "easy" but the scenarios are significantly different, we expect many major iterations relative to minor iterations. If the scenarios are similar, but each difficult, we expect few major iterations relative to minor iterations.

4.4 Stochastic Right-hand Sides

To simulate the occurrence of stochastic right-hand sides throughout the planning horizon of a dynamic problem, we modified the deter3 test problem to include stochastic demands on one node at each time period. The demands were uniformly distributed on an interval. Figure 5 summarizes the results for the range of bounds [L,U] shown.

The algorithm does not seem very sensitive to the occurrence of stochastic demands. It solves all the test problems in about the same time as in the case with deterministic demands. Only when the problems become close to being infeasible (i.e., demands being nearly as large as the total supplies) does the solution time increase substantially. Conversely, the problems are actually solved faster in cases with moderate stochastic demands than when no demands are present.

4.5 Solving Large-Scale Problems

To test the algorithm on large-scale problems, we modified the largest problem, deter3, by replicating the scenarios until there were 128, 512,1024 and 2048 scenarios, respectively. The scenarios were thus not all different. There is no reason to believe that the problems would have been substantially more difficult if this were the case.

The problems were run on an 8K and a 32K CM-2 with 32Kbytes of random-access

Scenarios	Equivalent NLP size	8K PEs		32K PEs	
		$\epsilon = 10^{-3}$	$\epsilon = 10^{-4}$	$\epsilon = 10^{-3}$	$\epsilon = 10^{-4}$
128	15488 x 42880	30.1	46.2	10.4	16.2
512	61952 x 171520	108.2	155.4	30.7	46.3
1024	123904 x 343040	210.8	326.5	57.3	86.3
2048	247808 x 686080	407.5	623.1	113.1	163.6

Table 4: Solving large-scale problems (solution times in seconds)

memory per processing element. Results are shown in Table 4, with a final tolerance of $\epsilon = 10^{-3}$ and $\epsilon = 10^{-4}$.

This experiment demonstrates the suitability of the algorithm for solving large-scale stochastic problems. The largest problem, 2048 scenarios, having a deterministic nonlinear equivalent of 247808 constraints and 686080 variables, was solved to the tightest tolerance in less than 3 minutes on the 32K CM-2. We also observe that the algorithm scales very effectively for larger problems on bigger machine sizes. For example, 512 scenarios are solved in 108 sec. on the 8K CM-2. Using a system with 32K processing elements we solve a problem with four times as many scenarios in almost the same time, 113 sec. On the largest CM-2 we have access to (32K processors with 32Kbytes of memory per processor) we can solve a problem with 8192 scenarios in approximately 11 minutes. On a maximally configured 64K CM-2 with 128 Kbytes of memory per processor we could potentially solve 64K scenarios in 45 minutes. Although such systems are available we do not have access to them.

5 Conclusion

We have in this study demonstrated the suitability of the row-action algorithm for solving quadratic stochastic network problems. The algorithm is well suited for fine-grained parallel implementations, making it ideal for implementation on massively parallel SIMD computers. The algorithm solves a range of realistic problems efficiently, and performs well even for very large-scale problems. However, we have identified cases involving extreme network characteristics (widely varying objective function coefficients and/or arc gains) where the algorithm's performance deteriorates.

The algorithm was implemented on a Connection Machine CM-2. The implementation can solve sparse problems, and still achieves a high computational rate, exceeding 0.5 GFLOPS on a 64K machine. It thus appears that row-action algorithms can be very effective on massively parallel computers, even though they may be less so in a serial environment. This enforces our prior experiences with transportation and multicommodity network problems.

The algorithm studied in this paper is restricted to solving quadratic problems. This is a

rather severe limitation. Nevertheless the algorithm presented here is the building block for solving general, linear programs. This can be achieved in the context of the proximal point algorithm of Rockafellar [1976], the proximal minimization with D-functions of Censor and Zenios [1989], or the nonlinear perturbations of Mangasarian and Meyer [1979]. This is the subject of a current study.

Acknowledgements. The authors would like to acknowledge the many useful discussions we have enjoyed with Professor Yair Censor during this project. We also thank J. Mesirov for constant encouragement. J. Mulvey and H. Vladimirou deserve thanks for providing us access to a number of their test problems. This project was completed while the authors were visiting Thinking Machines Corporation and (one of the authors) the Operations Research Center at MIT. 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] K.A. Ariyawansa and D. D. Hudson. Performance of a benchmark parallel implementation of the Van Slyke and Wets algorithm for two-stage stochastic programs on the Sequent/Balance. Technical report, Department of Pure and Applied Mathematics, Washington State University, Pullman, WA 99164-2930, 1989.

[2] K.A. Ariyawansa, D.C. Sorensen, and R.J. B. Wets. Parallel schemes to approximate values and subgradients of the resource function in certain stochastic programs. Technical report, Department of Pure and Applied Mathematics, Washington State University, Pullman, WA 99164-2930, 1990.

[3] E. M. L. Beale. On minimizing a convex function subject to linear inequalities. *J. Roy. Stat. Soc*, 17b:173–184, 1955.

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

[5] J. R. Birge. The value of the stochastic solution in stochastic linear programs with fixed recourse. *Mathematical Programming*, 24:314–325, 1982.

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

[7] Tamra J. Carpenter, Irvin J. Lustig, John M. Mulvey, and David F. Shanno. A primal-dual interior point method for convex separable nonlinear programs. Technical report SOR90-2, School of Engineering and Applied Science, Princeston University, April 1990.

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

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

[10] Y. Censor and S.A. Zenios. The proximal minimization algorithm with d-functions. Report 89-12-17, Decision Sciences Department, The Wharton School, University of Pennsylvania, Philadelphia, PA 19104, 1989.

[11] G. B. Dantzig. Linear programming under uncertainty. *Management Science*, 1:197–206, 1955.

[12] G. B. Dantzig, M. A. H. Dempster, and M. J. Kallio (ed.). Large-scale linear programming (volume 1). In *IIASA Collaborative Proceedings Series.* Laxenburg, Austria, 1981. CP-81-51.

[13] G.B. Dantzig. Planning under uncertainty using parallel computing. In *Annals of Operations Research*, volume 14, pages 1–17, 1985.

[14] George B. Dantzig and Peter W. Glynn. Parallel processors for planning under uncertainty. *Annals of Operations Research*, 22:1–21, 1990.

[15] 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.

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

[17] Gerd Infanger. Monte Carlo (importance) sampling within a Benders decomposition algorithm for stochastic linear programs. Technical report SOL 89-13R, Systems Optimization Laboratory, Department of Operations Research, Stanford University, Stanford, California 94305-4022, August 1966.

[18] O. L. Mangasarian and R. R. Meyer. Nonlinear perturbations on linear programs. *SIAM Journal on Control and Optimization*, 17:745–752, 1979.

[19] J.M. Mulvey and H. Vladimirou. Solving multistage stochastic networks: An application of scenario aggregation. Networks, 1990 (to appear).

[20] J.M. Mulvey and H. Vladimirou. Evaluation of a parallel hedging algorithm for stochastic network programming. In R. Sharda, B.L. Golden, E. Wasil, O. Balci, and W. Stewart, editors, *Impact of Recent Computer Advances on Operations Research*, 1989.

[21] 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.

[22] L. Qi. Forest iteration method for stochastic transportation problem. *Mathematical Programming Study 25*, pages 142–163, 1985.

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

[24] R.T. Rockafellar and R.J.-B. Wets. Scenarios and policy aggregation in optimization under uncertainty. Working paper wp–87–119, IIASA, Dec. 1987.

[25] A. Ruszczynski. A regularized decomposition method for minimizing a sum of polyhedral functions. *Mathematical Programming*, 35:309–333, 1986.

[26] A. Ruszczynski. Parallel decomposition of multistage stochastic programming problems. Working paper wp–88–094, IIASA, October. 1988.

[27] Andrzej Ruszczynski. Regularized decomposition and augmented Lagrangian decomposition for angular linear programming problems. *Aspiration Based Decision Support Systems*, 1989. A. Levandowski, A. P. Wierzbicki (Eds). Springer Verlag.

[28] R. Van Slyke and R. J. Wets. Programming under uncertainty and stochastic optimal control. *SIAM Journal on Control and Optimization*, 4:179–193, 1966.

[29] R. Van Slyke and R. J. Wets. L-shaped linear programs with applications to optimal control and stochastic programming. *SIAM Journal of Applied Mathematics*, 17:638–663, 1969.

[30] S. W. Wallace. Solving stochastic programs with network recourse. *Networks*, 16:295–317, 1986.

[31] S.W. Wallace. A two-stage stochastic facility location problem with time-dependent supply. Working paper, department of science and technology, Christian Michelsen Institute, Bergen, Norway, 1984.

[32] R. Wets. On parallel processor design for solving stochastic programs. Report wp–85–67, International Institute for Applied Systems Analysis, Laxenburg, Austria, Oct. 1985.

[33] R. J. Wets. The aggregation principle in scenario analysis and stochastic optimization. Working paper, Department of Mathematics, University of California, Davis, 1988.

[34] R. J. B. Wets. Stochastic programs with fixed resources: the equivalent deterministic problem. *SIAM Review*, 16:309–339, 1974.

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

[36] S.A. Zenios. On the fine-grain decomposition of multicommodity transportation problems. Technical report 90–09–07, Decision Sciences Department, The Wharton School, University of Pennsylvania, Philadelphia, PA 19104; October 1990.

[37] 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 19104, 1989.

[38] S.A. Zenios and J.M. Mulvey. A distributed algorithm for convex network optimization problems. *Parallel Computing*, 6:45–56, 1988.

[39] 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 19104, 1990.