NONLINEAR JACOBI AND ϵ-RELAXATION METHODS FOR PARALLEL NETWORK OPTIMIZATION

By

Armand A. Zakarian

A dissertation submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

(Computer Sciences)

at the

UNIVERSITY OF WISCONSIN – MADISON

1995

Abstract

In this thesis we develop an efficient decomposition method for large-scale convex cost multi-commodity network flow problems. The coupling constraints are moved to the objective function via augmented Lagrangian terms. A nonlinear Jacobi algorithm is then used to solve the resulting program in parallel in a fork-join manner. Several approaches to solving the coordination problem (corresponding to the join phase) are investigated. In particular, a coordination method is developed that combines excellent parallel efficiency and low communication overhead with a good empirical rate of convergence. Parallel implementations of the algorithm on a Thinking Machines CM-5 supercomputer and a cluster of workstations running PVM demonstrate that it is competitive or superior to the best known methods. We are able to solve linear multi-commodity problems with over 200,000 variables in less than 15 minutes.

The specific form of the non-strictly-convex subproblems (corresponding to the fork phase) motivates us to develop a relaxation method for separable convex network flow problems that is well-suited for problems with large variations in the magnitude of the nonlinear cost terms. The arcs are partitioned into two sets, one of which contains only arcs corresponding to strictly convex costs. The algorithm adjusts flows on the other arcs whenever possible, and terminates with primal-dual pairs that satisfy complementary slackness on the strictly convex arc set and ϵ-complementary slackness on the remaining arcs. An asynchronous parallel variant of the method is also developed. Computational results demonstrate that the method is significantly more efficient on quadratic ill-conditioned networks than existing methods, solving problems with several thousand nonlinear arcs in one second or less.

Acknowledgements

My deepest thanks go to my advisor Bob Meyer for his guidance, support and careful reading of this thesis. I would also like to thank the members of my Ph.D. examination committee Prof. Olvi Mangasarian, Prof. Michael Ferris, Prof. Anne Condon and Prof. John Strikwerda for carefully reviewing this thesis. Renato De Leone made many important suggestions in the early stages of this work. I would also like to acknowledge Golbon Zakeri, Ioannis Christou, and Spyros Kontogiorgis for numerous enjoyable and useful discussions.

This research was partially supported by the Air Force Office of Scientific Research Grant F49620-94-1-0036 and by the National Science Foundation Grants CDA-9024618 and CCR-9306807.

Contents

Chapter 1

Introduction

1.1 Overview

In this chapter we introduce the multi-commodity network flow problem and briefly discuss existing decomposition approaches that are amenable to parallelization.

In section 1.2 we formulate the problem under consideration and discuss some applications. In section 1.3 we review several approaches to decomposing the problem and the corresponding solution strategies. These decomposition methods are broadly classified into price-directive and resource-directive methods. The final section 1.4 introduces the decomposition approach taken in this thesis and briefly discusses the type of subproblem it generates.

1.2 Problem definition and applications

We consider the solution of the multi-commodity network flow problem (MC) using a parallel computer. MC is a special case of the general block-angular program:

$$
\begin{array}{llcl}
\underset{x}{\text{minimize}} & F(x_1, x_2, \ldots, x_K) & & \\
& A_1 x_1 & = & b_1 \\
& \quad A_2 x_2 & = & b_2 \\
& \qquad \ddots & & \\
& \qquad\qquad A_K x_K & = & b_K \\
& D_1 x_1 + D_2 x_2 + \ldots + D_K x_K & \leq & d \\
& 0 \leq x_i \leq u_i, \quad i = 1, \ldots, K & &
\end{array} \tag{1}
$$

Here A_i are $m_i \times n_i$ node-arc incidence matrices, D_i are $m_0 \times n_i$ matrices, x_i and u_i are elements of R^{n_i}, b_i are elements of R^{m_i} and d is in R^{m_0}. We define

$$\mathcal{B}_i := \{y \mid A_i y = b_i,\ 0 \leq y \leq u_i\}$$

and $\mathcal{B} := \mathcal{B}_1 \times \ldots \times \mathcal{B}_K$. We further assume that the non-zero columns of each D_i are a subset of the columns of the identity matrix I_{m_0}. The function $F : R^{n_1+n_2+\ldots+n_K} \mapsto R$ is convex. When $F(x_1, x_2, \ldots, x_K)$ is block separable we will use the expression $\sum_{i=1}^{K} f_i(x_i)$ instead. We will be primarily interested in the linear MC, i.e. $f_i(x_i) \equiv c_i^T x_i$. We denote the concatenation of vectors $x_1, x_2, \ldots, x_K$ by x and the matrix $[D_1 D_2 \ldots D_K]$ by $\mathcal{D}$.

We assume that problem (1) is solvable.

In MC the node-arc incidence matrices $A_i, i = 1, \ldots, K$ are often the same and represent a common network shared by K commodities, each subject to separate supply/demand constraints. The coupling constraints $\mathcal{D}x \leq d$ represent sharing of a common resource d by the commodities.

In multi-commodity network flow models commodities may correspond to separate physical goods or alternatively, to separate origin-destination pairs for a single physical good. The first model is typical for applications such as railroad transportation networks or computer networks. The second type frequently arises in communication and distribution/transportation networks. Other applications of MC include optimal deployment of resources (Kaplan 1973), VLSI circuit layout (Korte 1988) and multi-product multi-stage production-inventory planning (Evans 1977).

1.3 Overview of parallel decomposition methods

In this section we briefly describe several approaches to decomposing MC for parallel solution. The first four methods share some price-directive elements. We have chosen to present them based on the fact that they have been successfully used to solve large instances of MC on parallel computers. The corresponding numerical results will be used as a basis for comparison with the method developed in this thesis and will appear in Chapter 4. To present a more complete picture of parallel decomposition methods, we discuss in the last subsection some resource-directive based approaches.

1.3.1 The Dantzig-Wolfe method

Dantzig-Wolfe decomposition (Dantzig & Wolfe 1960) is a classical price-directive solution approach for linear block-angular programs. Let X_i and S_i be matrices whose columns are the extreme points and the extreme rays of $\mathcal{B}_i$, respectively. Then

$$x_i \in \mathcal{B}_i \iff x_i = X_i\alpha_i + S_i\beta_i, \;\; e^T\alpha_i = 1, \;\; \alpha_i \geq 0, \;\; \beta_i \geq 0$$

and consequently an equivalent formulation of MC is

$$\begin{array}{ll} \underset{\alpha,\beta}{\text{minimize}} & \sum_{i=1}^K c_i^T X_i\alpha_i + \sum_{i=1}^K c_i^T S_i\beta_i \\ & \sum_{i=1}^K D_iX_i\alpha_i + \sum_{i=1}^K D_iS_i\beta_i + s = d \\ \text{subject to} & e^T\alpha_i = 1, \;\; i = 1, \ldots, K \\ & \alpha \geq 0, \; \beta \geq 0, \; s \geq 0 \end{array} \tag{2}$$

where s is a vector of slack variables.

Dantzig-Wolfe uses a column generation technique to solve (2). Pricing in (2) amounts to solving the subproblems

$$\underset{x_i \in \mathcal{B}_i}{\text{minimize}} \; (c_i - D_i^T p)^T x_i - q_i$$

by an extreme point algorithm. Here p and q_i are the current values of the dual variables in (2). If a subproblem j has a negative optimal value and optimal solution point $\hat{x}_j$ then the component of α_j corresponding to $\hat{x}_j$ is a candidate for basis entry in (2) (similarly for unbounded subproblems and components of β_j). Once an entering variable is identified, the corresponding column is generated, updated and a pivot in (2) (an update of the basis inverse) is performed.

A fork-join parallel implementation of the method where one processor is dedicated to solving the master problem and another K processors solve the subproblems independently is described in (De Leone, Meyer, Kontogiorgis, Zakarian & Zakeri 1994). Unfortunately the master program may become relatively dense resulting in master solution times dominating subproblem solution times and ultimately, decreased parallel efficiency.

1.3.2 Linear-quadratic penalty algorithm

Zenios, Pınar & Dembo (1991) develop a decomposition for MC based on penalizing the violation of the coupling constraints via a smoothed exact penalty function. The smooth penalty version of MC can be written in the form

$$\underset{x \in \mathcal{B}}{\text{minimize}} \quad \Phi_{\mu,\epsilon} = \sum_{i=1}^{K} f_i(x_i) + \mu \sum_{j=1}^{m_0} \phi(\epsilon, y_j) \tag{3}$$

where $y_j = (\mathcal{D}x - d)_j$ and the linear-quadratic penalty function ϕ is given by

$$\phi(\epsilon, t) := \begin{cases} 0 & \text{if } t \leq 0 \\ \frac{t^2}{2\epsilon} & \text{if } 0 \leq t \leq \epsilon \\ t - \frac{\epsilon}{2} & \text{if } t \geq \epsilon \end{cases}$$

The linear-quadratic penalty function is a “smoothing” function and is further discussed by Chen & Mangasarian (1994). It is a convex continuously differentiable function and thus allows solution of (3) using smooth optimization techniques. The linearity of ϕ for $t \geq \epsilon$ allows solutions of (3) to approximate solutions of MC arbitrarily well as $\epsilon \to 0$ with the penalty parameter μ remaining bounded.

Problem (3) is non-separable. To make it amenable to parallel computation, it is linearized thus generating K independent linear network flow problems. Descent directions generated by these subproblems are communicated to a master problem based on simplicial decomposition. The master (which tends to dominate total solution time) involves heavy dense linear algebra computations and consequently the linear-quadratic penalty method benefits greatly when implemented for a vector supercomputer.

1.3.3 Barrier methods

Schultz & Meyer (1991) consider a logarithmic barrier version of MC. Given a current iterate $x^t \in \mathcal{B}$ the next iterate x^{t+1} is taken as the solution of the shifted barrier problem

$$\underset{x \in \mathcal{B}}{\text{minimize}}\, F(x, \tau, \theta) := \sum_{i=1}^{K} f_i(x_i) - \tau \sum_{j=1}^{m_0} \log{(\theta - \mathcal{D}x)_j} \tag{4}$$

where the barrier θ is such that $\mathcal{D}x < \theta$.

The solution method consists of three phases. In the first phase, a point $x^0 \in \mathcal{B}$ is found. The second phase seeks to find a feasible solution of MC by holding τ constant and shifting the barrier θ towards d:

$$\theta_j^{t+1} = \begin{cases} d_j & \text{if } (\mathcal{D}x^t - d)_j < 0 \\ \lambda(\mathcal{D}x^t)_j + (1-\lambda)\theta_j^t & \text{otherwise} \end{cases}$$

The final phase tries to approximate optimal solutions of MC by keeping $\theta = d$ and reducing the barrier term by letting $\tau \to 0$.

To decompose problem (4) by commodity, Schultz & Meyer linearize (4) around x^t. The resulting subproblems (which also employ trust region constraints) generate descent directions for each block in parallel. These directions are then used by a multi-dimensional line-search master procedure to find approximate solutions of (4).

The method is further developed by Zakeri (1995) who proposes replacing the master problem of (Schultz & Meyer 1991) by a collection of (smaller) coordinating problems that can be solved simultaneously thus improving the parallel efficiency of the original method.

1.3.4 Method of alternating directions

Kontogiorgis (1994) specializes the Alternating Directions method originally of (Glowinski & Marrocco 1975) for the parallel solution of MC. The Alternating Directions method seeks to solve the non-smooth programming problem

$$\begin{array}{ll} \underset{x,z}{\text{minimize}} & G_1(x) + G_2(z) \\ \text{subject to} & Ax + b = Bz \end{array} \tag{5}$$

where G_1 and G_2 are extended-valued real proper convex functions by minimizing the Augmented Lagrangian

$$L_\lambda(x, z, p) := G_1(x) + G_2(z) + p^T(Ax + b - Bz) + \frac{\lambda}{2}\|Ax + b - Bz\|_2^2$$

repeatedly in a block Gauss-Seidel fashion:

$$x^{t+1} \in \underset{x}{\operatorname{argmin}}\, L_\lambda(x, z^t, p^t)$$

$$z^{t+1} \in \operatorname*{argmin}_z L_\lambda(x^{t+1}, z, p^t)$$
$$p^{t+1} = p^t + \lambda \nabla_p L_\lambda(x^{t+1}, z^{t+1}, p^t)$$

The Activity-and-Resource Proximization splitting of (Kontogiorgis 1994) casts MC in the form of (5) by introducing variables $\tilde{d}_1, \ldots, \tilde{d}_K$ representing the allocation of the common resource d to the commodities and defining

$$h_i(x_i) := \begin{cases} f_i(x_i) & \text{if } A_i x_i = b_i \text{ and } 0 \leq x_i \leq u_i \\ +\infty & \text{otherwise,} \end{cases}$$

$$G_1(x_1, \ldots, x_K, \tilde{d}_1, \ldots, \tilde{d}_K) := \begin{cases} \sum_{i=1}^K h_i(x_i) & \text{if } D_i x_i \leq \tilde{d}_i \ \forall i \\ +\infty & \text{otherwise,} \end{cases}$$

and

$$G_2(y_1, \ldots, y_K, d_1, \ldots, d_K) := \begin{cases} 0 & \text{if } \sum_{i=1}^K d_i = d \\ +\infty & \text{otherwise.} \end{cases}$$

The matrices A and B and the vector b in (5) are taken to be identity matrices and zero, respectively. The linear constraints of (5) thus become simply $x_i = y_i,\ \tilde{d}_i = d_i\ \forall i$.

In (Kontogiorgis 1994) two other splittings for MC are introduced—the Resource Proximization splitting and the Activity Proximization splitting. They turn out to be in general computationally superior to ARP mainly due to the smaller dimension of the quadratic subproblems solved at each iteration of the Alternating Directions algorithm.

1.3.5 Resource-Directive decomposition

In resource-directive decomposition the common resource d is split among the commodities. Then the problem is decomposed into K independent min-cost network flow problems that have the same network constraints as the original MC but with upper bounds on the flow variables adjusted to account for the portions of the common resource assigned to each commodity. Information from optimizing these subproblems is then used to adjust the splitting of d among the commodities. Consider the problems

$$z_i(y_i) := \min \{ f_i(x_i) \mid x_i \in \mathcal{B}_i,\ D_i x_i \le y_i \} \tag{6}$$

for $i = 1, \ldots, K$ and let

$$\Phi(y_1, y_2, \ldots, y_K) := \sum_{i=1}^{K} z_i(y_i)$$

Then MC is equivalent to

$$\begin{array}{ll} \underset{y_1, y_2, \ldots, y_K}{\text{minimize}} & \Phi(y_1, y_2, \ldots, y_K) \\ \text{subject to} & \begin{array}{c} \sum_{i=1}^{K} y_i = d \\ y_i \ge 0,\ i = 1, \ldots, K \end{array} \end{array} \tag{7}$$

Since Φ is, in general, a convex non-differentiable function, the above problem needs to be attacked using non-smooth optimization techniques.

Bundle methods

De Leone, Gaudioso & Monaco (1992) develop a bundle-type method for (7). In the case of the linear problem it is possible to show that if $\hat{x}_i$ is an ϵ_i-optimal solution point and λ_i are the corresponding multipliers for the constraints $D_i x_i \leq y_i$ in (6) for each i then $g = (\lambda_1, \lambda_2, \ldots, \lambda_K)$ is an ϵ-subgradient of $\Phi(y_1, y_2, \ldots, y_K)$ where $\epsilon = \sum_{i=1}^K \epsilon_i$. Also $h = \sum_{i=1}^K f_i(\hat{x}_i)$ is within ϵ of the objective function value $\Phi(y_1, y_2, \ldots, y_K)$.

The algorithm maintains a "bundle" of information triplets (y^l, h^l, g^l) computed at previous time-steps $l \in I$. A tentative change for the resource splitting y^t at the current time-step t is then given by the optimal solution point s of

$$\begin{array}{ll} \underset{v,s}{\text{minimize}} & v + \frac{1}{2\mu}\|s\|^2 \\ \text{subject to} & \begin{array}{c} v \geq {g^t}^T s - \epsilon^t \\ v \geq {g^l}^T s - \xi^l, \ l \in I \end{array} \end{array} \tag{8}$$

where $\mu > 0$ and

$$\xi^l = 2\epsilon^l + h^t - h^l - {g^l}^T (y^t - y^l).$$

If the tentative new iterate $y^{t+1/2} := y^t + s$ provides sufficient decrease in the value of Φ, then it becomes the next iterate y^{t+1}. Otherwise, $(y^{t+1/2}, h^{t+1/2}, g^{t+1/2})$ is added to the bundle I and (8) is re-optimized. Termination occurs when the optimal solution point v of (8) becomes small in magnitude.

A bundle approach has also been used by Medhi (1987) to optimize the Rockafellar dual

$$\max_{p \leq 0} \left\{ p^T d + \sum_{i=1}^K \min_{x_i \in \mathcal{B}_i} \left\{ f_i(x_i) - p^T D_i x_i \right\} \right\}$$

of MC in parallel using the fact that the inner minimization is separable by block.

1.4 The augmented Lagrangian decomposition

Augmented Lagrangian decomposition has been successfully used (Ruszczyński 1993*b*) for solving large-scale block angular programs arising in the deterministic formulation of multi-stage stochastic programs.

The T-stage stochastic program has the form:

$$\min_{x_1} \left\{ c_1 x_1 + E_{\xi_2} \left[\min_{x_2} \left(\mathbf{c}_2 x_2 + E_{\xi_3|\xi_2} (\min_{x_3} \mathbf{c}_3 x_3 + \cdots + E_{\xi_T|\xi_2,\ldots,\xi_{T-1}} \min_{x_T} \mathbf{c}_T x_T) \right) \right] \right\}$$

$$\begin{array}{lllllllll} \text{subject to} & A_1 x_1 & & & & & & = & b_1 \\ & \mathbf{B}_2 x_1 & + & \mathbf{A}_2 x_2 & & & & = & \mathbf{b}_2 \\ & & & \mathbf{B}_3 x_2 & + & \mathbf{A}_3 x_3 & & = & \mathbf{b}_3 \\ & & & & & \ddots & & & \vdots \\ & & & & & \mathbf{B}_T x_{T-1} & + \; \mathbf{A}_T x_T & = & \mathbf{b}_T \\ & & & 0 \le x_t \le u_t, & t = 1, \ldots T, & & & & \end{array} \tag{9}$$

where

$$\xi_t = (\mathbf{A}_t, \mathbf{B}_t, \mathbf{b}_t, \mathbf{c}_t), \quad t = 2, \ldots, T$$

are random variables drawn from some probability space, E_ξ denotes mathematical expectation with respect to ξ and x_t are stochastic decision variables.

In the deterministic formulation of (9) only a finite number of possible realizations of each random variable ξ_t are considered. A scenario $s \in \Omega$ is a realization of the random variable $(\xi_2, \xi_3, \ldots, \xi_T)$. The sequential decision process evident in (9) can now be modeled by a scenario tree whose nodes at level $t \geq 2$ correspond to the possible realizations of ξ_t and whose root corresponds to the (deterministic) data (A_1, b_1, c_1). Every scenario in Ω can be associated with a specific path from the root to a leaf node in the scenario tree.

We now introduce the split-variable deterministic formulation of (9) (Rockafellar & Wets 1991). For a given scenario s we consider the scenario subproblem:

$$\begin{array}{lllllll}
\underset{x_1^s, x_2^s, \ldots, x_T^s}{\text{minimize}} & c_1^s x_1^s + c_2^s x_2^s + \ldots + c_T^s x_T^s & & & & & \\
& A_1 x_1^s & & & & = & b_1 \\
& B_2^s x_1^s & + \; A_2^s x_2^s & & & = & b_2^s \\
\text{subject to} & & B_3^s x_2^s & + \; A_3^s x_3^s & & = & b_3^s \\
& & & \ddots & & & \vdots \\
& & & & B_T^s x_{T-1}^s \; + \; A_T^s x_T^s & = & b_T^s \\
& & 0 \le x_t^s \le u_t, \quad t = 1, \ldots T. & & & &
\end{array} \tag{10}$$

Here the decision variables x_t^s have taken the place of the stochastic variables x_t. We denote the objective function and feasible set of (10) by $f^s(x^s)$ and $\mathcal{B}^s$, respectively.

The scenario subproblems introduced so far are independent for different scenarios. The non-anticipativity constraints serve to link the scenario subproblems together. These can be stated thusly: $x_t^s = x_t^{s+1}$ if the node at level t in the scenario tree is common for scenarios s and $s+1$. Given the probability

p^s of scenario s the original problem can be stated as follows:

$$\begin{array}{ll} \text{minimize} & \sum_{s\in\Omega} p^s f^s(x^s) \\ \text{subject to} & \begin{array}{l} x^s \in \mathcal{B}^s, \quad s \in \Omega \\ \sum_{s\in\Omega} D^s x^s = 0, \end{array} \end{array} \tag{11}$$

where $\sum_{s\in\Omega} D^s x^s = 0$ is a matrix representation of the non-anticipativity constraints.

The independence of the scenario subproblems (10) motivates decomposition procedures for the multistage program that relax the non-anticipativity constraints. Rosa & Ruszczyński (1994) have successfully solved large multistage programs using an augmented Lagrangian formulation of (11):

$$\Lambda(x,\pi) = \sum_{s\in\Omega} p^s f^s(x^s) - \langle \pi, \sum_{s\in\Omega} D^s x^s \rangle + \frac{\lambda}{2} \| \sum_{s\in\Omega} D^s x^s \|_2^2$$

The following method of multipliers is used to generate a sequence converging to the solution point of (11):

Step 0. Let a set of multipliers π^0 be given. Set $k = 0$.

Step 1. Let $(x^{1,k+1}, x^{2,k+1}, \ldots, x^{|\Omega|,k+1})$ be the solution point of:

$$\begin{array}{ll} \underset{x}{\text{minimize}} & \Lambda(x, \pi^k) \\ \text{subject to} & x^s \in \mathcal{B}^s, \quad s \in \Omega \end{array} \tag{12}$$

Step 2. Update the multipliers using:

$$\pi^{k+1} = \pi^k - \lambda(\sum_{s\in\Omega} D^s x^{s,k+1})$$

Step 3. Update the penalty parameter λ if necessary, set $k = k + 1$ and return to **Step 1**.

The constraint set of problem (12) is a Cartesian product of constraint sets but its objective function is non-separable. An application of the nonlinear Jacobi algorithm allows for a parallel solution procedure. Let

$$\Lambda^r(y^r, x, \pi) :=$$
$$p^r f^r(y^r) + \sum_{s \in \Omega \setminus r} p^s f^s(x^s) - \langle \pi, D^r y^r + \sum_{s \in \Omega \setminus r} D^s x^s \rangle + \frac{\lambda}{2} \| D^r y^r + \sum_{s \in \Omega \setminus r} D^s x^s \|_2^2$$

The nonlinear Jacobi algorithm proceeds by first computing in parallel

$$\tilde{x}^r = \operatorname*{argmin}_{y^r \in \mathcal{B}^r} \Lambda^r(y^r, x, \pi), \quad r \in \Omega \tag{13}$$

and then updating the current iterate x by

$$x^r \leftarrow x^r + \tau(\tilde{x}^r - x^r), \quad r \in \Omega, \tag{14}$$

where τ is a chosen constant. The convergence theory for the nonlinear Jacobi algorithm developed in (Ruszczyński 1993a) (and further refined in Chapter 3) allows the value of τ to be arbitrarily close to $1/N$ where N is the maximum number of scenarios linked by any coupling constraint minus one. In the split-variable formulation of the multistage program the non-anticipativity constraints are defined so as to have $N = 1$ and hence τ can be taken close to 1. This results in a relatively fast convergence of the method.

In Chapter 3 we consider an application of the nonlinear Jacobi algorithm to the augmented Lagrangian formulation of MC. One possible augmented Lagrangian (here we have returned to our usual notation) is:

$$L_\lambda(x,p) = F(x) + p^T(\mathcal{D}x - d) + \frac{\lambda}{2}\|\hat{\mathcal{D}}x - \hat{d}\|_2^2 \tag{15}$$

Here $\hat{\mathcal{D}}$ and $\hat{d}$ denote a subset of the rows of $\mathcal{D}$ and d corresponding to the current "active set" of coupling constraints. The "active set" contains constraints that either have positive multipliers or are satisfied as equality or violated. A possible motivation for our choice of an "active set" strategy is the desire to add to the objective function as few non-separable terms as possible.

The coordination step (14) may not be appropriate for MC since τ must be less than $\frac{1}{K-1}$ and convergence will be slow. We consider alternative coordination mechanisms and investigate the inherent trade-offs between complexity of coordination and speed of convergence of the nonlinear Jacobi algorithm in Chapter 3.

1.4.1 Subproblems arising in the augmented Lagrangian decomposition

We have discussed in this chapter several methods for decomposing MC suitable for parallel computation. We are also interested in developing an efficient algorithm for the type of subproblem that arises in the decomposition. In the case of the nonlinear Jacobi method applied to the augmented Lagrangian formulation of MC, the network flow subproblem analogous to (13) has the following form:

$$\begin{array}{ll} \underset{x}{\text{minimize}} & \sum_{l\in\bar{\mathcal{A}}} g_l(x_l) + \sum_{l\in\mathcal{A}} \{g_l(x_l) + \frac{\lambda}{2}x_l^2\} \\ \text{subject to} & \begin{array}{c} Ax = b \\ 0 \le x \le u \end{array} \end{array} \tag{16}$$

$\mathcal{A}$ denotes the set of arcs that participate in the "active" set of coupling constraints, i.e. have nonzero coefficients in $\hat{D}_i$. $\bar{\mathcal{A}}$ is the complement of $\mathcal{A}$. We have dropped any reference to the block index r but have written the objective function in terms of the components of x to emphasize its separability. The functions g_l capture both the original objective function and the linear terms from the augmented Lagrangian.

In the case of the linear MC (16) is a min-cost problem that has a mixture of linear and quadratic arc cost functions. Even when MC is nonlinear (16) is likely to be ill-conditioned due to the need for increasing the penalty parameter λ in the course of the augmented Lagrangian algorithm and the resulting differential in the curvature of the cost functions in $\mathcal{A}$ vs. that in $\bar{\mathcal{A}}$. In the following Chapter 2 we develop a new algorithm suitable for this type of convex network flow problem. Parallel versions of this algorithm are also developed in Appendix A.

Chapter 2

The convex min-cost network flow problem

2.1 Overview

In this chapter we present a method for solving the convex min-cost network flow problems that arise in the decomposition of the multi-commodity network flow problem.

In section 2.2 we define the problem under consideration and mention some applications. In section 2.3 the proposed algorithm is described. The section also provides the formal convergence proof, a discussion of computational refinements and the specialization of the algorithm to linear-quadratic problems. The final section 2.4 briefly describes a method for solving the same type of problem based on the network simplex method. Computational comparison of the two approaches will be presented in Chapter 4.

2.2 Problem definition and motivation

Given a directed graph $\mathcal{G}(\mathcal{N}, \mathcal{A})$ with n nodes and m arcs, the minimum–cost network flow problem that we consider is:

$$\begin{aligned} &\text{minimize} && \sum_{(i,j)\in\mathcal{A}} f_{ij}(x_{ij}) \\ &\text{subject to} && \sum_{j\in\delta_+(i)} x_{ij} - \sum_{j\in\delta_-(i)} x_{ji} = b_i, \quad i \in \mathcal{N} \end{aligned} \tag{17}$$

where for each node i, $\delta_+(i)$ is the subset of nodes j such that $(i,j) \in \mathcal{A}$ and $\delta_-(i)$ is the subset of nodes j such that $(j,i) \in \mathcal{A}$.

Networks with nonlinear objectives arise in a variety of applications including electrical networks (Hu 1966), pipeline networks (Collins, Cooper, Helgason, Kennington & LeBlanc 1978), urban traffic flow (Magnanti 1984) and communications networks (Monma & Segal 1982).

We make the following assumptions with regard to the functions f_{ij}.

Assumption 2.2.1 *Each f_{ij} is a closed proper convex function and the intersection of* $\operatorname{dom}\left(\sum_{(i,j)\in\mathcal{A}} f_{ij}\right)$ *with the set of vectors satisfying the constraints of (17) is nonempty.*

Assumption 2.2.2 *The conjugate functions*

$$f_{ij}^*(t_{ij}) := \sup_{x_{ij}} \{t_{ij}x_{ij} - f_{ij}(x_{ij})\}$$

are finite-valued.

Assumption 2.2.1 guarantees that (17) is feasible. Assumption 2.2.2 implies that the objective function of (17) has bounded level sets and together with

Assumption 2.2.1 guarantees that an optimal solution exists. A linear cost function must have a bounded domain in order to satisfy Assumption 2.2.2.

Often the functions f_{ij} imply flow capacity constraints:

$$f_{ij}(x_{ij}) := \begin{cases} \bar{f}_{ij}(x_{ij}) & \text{if } l_{ij} \leq x_{ij} \leq u_{ij} \\ +\infty & \text{otherwise} \end{cases} \tag{18}$$

Here $\bar{f}_{ij}$ is a real-valued convex function and $l_{ij} \leq u_{ij}$ are the (finite) lower and upper bounds on the flow on arc $(i, j) \in \mathcal{A}$. In this case Assumption 2.2.2 is trivially satisfied.

We denote the left and right derivative of f_{ij} at $x_{ij} \in \text{dom} f_{ij}$ by $f_{ij}^-(x_{ij})$ and $f_{ij}^+(x_{ij})$, respectively. For points x_{ij} to the left of $\text{dom} f_{ij}$ we set $f_{ij}^-(x_{ij}) = f_{ij}^+(x_{ij}) = -\infty$ and for points x_{ij} to the right of $\text{dom} f_{ij}$ we set $f_{ij}^-(x_{ij}) = f_{ij}^+(x_{ij}) = +\infty$. Theorem 24.1 in (Rockafellar 1970) states that f_{ij}^- and f_{ij}^+ are nondecreasing functions on R, finite on $\text{int}\,\text{dom} f_{ij}$ and

$$f_{ij}^+(z_{ij}) \leq f_{ij}^-(x_{ij}) \leq f_{ij}^+(x_{ij}) \leq f_{ij}^-(y_{ij})$$

whenever $z_{ij} < x_{ij} < y_{ij}$.

We make an additional assumption regarding the functions f_{ij}:

Assumption 2.2.3 $f_{ij}^-(x_{ij}) < +\infty$ *and* $f_{ij}^+(x_{ij}) > -\infty$ *for all* $x_{ij} \in \text{dom} f_{ij}$.

Assumption 2.2.3 guarantees that the feasible solutions of (17) are *regularly feasible* ((Rockafellar 1984), Section 8D). Together with the first two assumptions it also guarantees the existence of dual optimal solutions ((Rockafellar 1984), Section 8L).

We say that a pair (x, π) satisfies complementary slackness (CS) on an arc $(i, j) \in \mathcal{A}$ if

$$f_{ij}^{-}(x_{ij}) \leq \pi_i - \pi_j \leq f_{ij}^{+}(x_{ij}) \tag{19}$$

For arbitrary vector π and f_{ij} strictly convex satisfying Assumption 2.2.2, there exists a unique x_{ij} satisfying (19). A vector x is optimal for (17) if it satisfies the flow conservation constraints and together with some π satisfies CS on every arc in $\mathcal{A}$.

A pair (x, π) satisfies ϵ-complementary slackness (ϵ-CS) on an arc $(i, j) \in \mathcal{A}$ if

$$f_{ij}^{-}(x_{ij}) - \epsilon \leq \pi_i - \pi_j \leq f_{ij}^{+}(x_{ij}) + \epsilon \tag{20}$$

We will see below that ϵ-complementary slackness corresponds to near optimality for small ϵ.

A statement of the basics of the algorithm is given in the following subsection. Subsection 2.3.2 provides a convergence analysis and a quantitative interpretation of the significance of ϵ-complementary slackness. Some computational refinements of the basic method are presented in subsection 2.3.3, followed by a specialization of the algorithm to the mixed linear-quadratic case. Finally, parallel versions of the algorithm are discussed in Appendix A.

2.3 The ϵ-relaxation algorithm

2.3.1 Statement of the algorithm

We state the algorithm in its simplest form in this section. A brief discussion is given of the main ideas, followed by a formal statement of the method.

The algorithm computes an approximate solution of problem (17) in the sense that it generates a sequence of iterates converging to a feasible solution satisfying ϵ-CS. More precisely, we assume that the set $\mathcal{A}$ has been partitioned into two sets $\mathcal{A}_0$ and $\mathcal{A}_1$ such that only arcs with strictly convex cost functions are in $\mathcal{A}_1$, and we show that in the limit the primal-dual pair produced by the algorithm will satisfy CS on each arc in $\mathcal{A}_1$ and ϵ-CS on each arc in $\mathcal{A}_0$. This approach differs from that of Bertsekas, Hosein & Tseng (1987) in that it divides the arcs into two sets and focuses on changing single node prices and flows on arcs incident to single nodes.

In **Step 1** below, a node i with positive surplus is identified. Steps 2–4 are then repeatedly applied until the surplus at this node is driven to 0 (which is achievable in a finite number of steps, see discussion following algorithm). The algorithm first tries to "push" this surplus to adjacent nodes using only arcs in $\mathcal{A}_0$ (**Step 2** and **3**). If unsuccessful, it then tries to raise the price (**Step 4**) of node i sufficiently high so that the surplus can be "pushed away" using the incident arcs in $\mathcal{A}_1$ but not so high as to violate ϵ-CS on the arcs in $\mathcal{A}_0$. We assume that ϵ and ϵ' are two given numbers such that $0 \leq \epsilon' < \frac{\epsilon}{2}$.

Algorithm 1

Step 0 Start with (x, π) satisfying CS on $\mathcal{A}_1$ and ϵ-CS on $\mathcal{A}_0$.

Step 1 Select node i with positive surplus. If there isn't such node, stop.

Step 2 Increase flow on outgoing arc in $\mathcal{A}_0$ with negative "reduced cost" maintaining ϵ'-CS. If surplus$_i$ is exhausted, go to Step 1.

Step 3 Decrease flow on incoming arc in $\mathcal{A}_0$ with positive "reduced cost" maintaining ϵ'-CS. If surplus$_i$ is exhausted, go to Step 1.

Step 4 Increase π_i and simultaneously adjust flows on incident arcs in $\mathcal{A}_1$ (maintaining CS) until surplus$_i$ $= 0$ or ϵ-CS on $\mathcal{A}_0$ is violated. If surplus$_i = 0$ go to Step 1, else return to Step 2.

We now present the steps of the algorithm in detail.

Step 0 [Initialization] Start with (x, π) satisfying ϵ-CS on all arcs in $\mathcal{A}_0$ and CS on all arcs in $\mathcal{A}_1$. Set $k = 0$.

Step 1 [Choose node with positive surplus] Choose a node i such that

$$s_i := b_i - \sum_{j \in \delta_+(i)} x_{ij} + \sum_{j \in \delta_-(i)} x_{ji} > 0$$

and increment k. If no such node exists, STOP.

Step 2 [Increase flow on outgoing arc in $\mathcal{A}_0$] If there is an arc $(i,j) \in \mathcal{A}_0$ such that

$$f_{ij}^{+}(x_{ij}) - (\pi_i - \pi_j) \leq -\frac{\epsilon}{2}$$

then change the flow on arc (i,j),

$$x_{ij} \leftarrow x_{ij} + \min\{s_i,\ \delta\}$$

where $\delta > 0$ is the largest flow increase allowing ϵ'-CS to be satisfied on arc (i,j):

$$f_{ij}^{-}(x_{ij} + \delta) - \epsilon' \leq \pi_i - \pi_j \leq f_{ij}^{+}(x_{ij} + \delta) + \epsilon'$$

The flow on the remaining arcs and the dual costs remain unchanged. Update s_i and if its new value is 0 go to Step 1 else return to Step 2.

Step 3 [Decrease flow on incoming arc in $\mathcal{A}_0$] If there is an arc $(j,i) \in \mathcal{A}_0$ such that

$$f_{ji}^{-}(x_{ji}) - (\pi_j - \pi_i) \geq \frac{\epsilon}{2}$$

then change the flow on arc (j,i),

$$x_{ji} \leftarrow x_{ji} - \min\{s_i,\ \delta\}$$

where $\delta > 0$ is the least flow decrease allowing ϵ'-CS to be satisfied on arc (j,i):

$$f_{ji}^{-}(x_{ji} - \delta) - \epsilon' \leq \pi_j - \pi_i \leq f_{ji}^{+}(x_{ji} - \delta) + \epsilon'$$

The flow on the remaining arcs and the dual costs remain unchanged. Update s_i and if its new value is 0 go to Step 1 else return to Step 3.

Step 4 [Raise price and change flows in $\mathcal{A}_1$] Compute

$$\Delta_1 := \min_{\{j\in\delta_+(i)\,|\,(i,j)\in\mathcal{A}_0\}} \left\{ f_{ij}^+(x_{ij}) - (\pi_i - \pi_j) \right\} \geq -\frac{\epsilon}{2}$$

$$\Delta_2 := \min_{\{j\in\delta_-(i)\,|\,(j,i)\in\mathcal{A}_0\}} \left\{ -\left(f_{ji}^-(x_{ji}) - (\pi_j - \pi_i) \right) \right\} \geq -\frac{\epsilon}{2}$$

For each γ and for each arc $(i,j) \in \mathcal{A}_1$ let $x_{ij}(\gamma)$ be the flow such that

$$f_{ij}^-(x_{ij}(\gamma)) \leq \pi_i + \gamma - \pi_j \leq f_{ij}^+(x_{ij}(\gamma))$$

Similarly for each arc $(j,i) \in \mathcal{A}_1$ let $x_{ji}(\gamma)$ be the flow such that

$$f_{ji}^-(x_{ji}(\gamma)) \leq \pi_j - \pi_i - \gamma \leq f_{ji}^+(x_{ji}(\gamma))$$

and let $\Delta_3 > 0$ be the value of γ such that

$$b_i - \sum_{(i,j)\in\mathcal{A}_1} x_{ij}(\gamma) - \sum_{(i,j)\in\mathcal{A}_0} x_{ij} + \sum_{(j,i)\in\mathcal{A}_1} x_{ji}(\gamma) + \sum_{(j,i)\in\mathcal{A}_0} x_{ji} = 0$$

(where $\Delta_3 = +\infty$ if a solution does not exist).

Define:

$$\Delta := \min\{\Delta_1 + \epsilon, \Delta_2 + \epsilon, \Delta_3\}$$

Change the price on node i:

$$\pi_i \leftarrow \pi_i + \Delta$$

and adjust the flow on the incident arcs in $\mathcal{A}_1$ so that CS is satisfied. The flow on all other arcs and the prices of all remaining nodes remains unchanged. Update s_i and if its new value is 0 go to Step 1 else return to Step 2.

We now verify that **Step 2** maintains ϵ-CS on arc (i, j) (this can be done similarly for **Step 3**). Let x'_{ij} be the updated value of the flow on (i, j). We have

$$f^-_{ij}(x'_{ij}) - \epsilon \leq f^-_{ij}(x_{ij} + \delta) - \epsilon \leq f^-_{ij}(x_{ij} + \delta) - \epsilon' \leq \pi_i - \pi_j$$

and

$$f^+_{ij}(x'_{ij}) + \epsilon \geq f^+_{ij}(x_{ij}) + \epsilon \geq \pi_i - \pi_j$$

To see that $\Delta > 0$ in **Step 4**, note that $\Delta_1 > -\frac{\epsilon}{2}$ and $\Delta_2 > -\frac{\epsilon}{2}$ since no eligible arcs could be found in **Step 2** and **Step 3**, respectively. Also, $\Delta_3 > 0$ since we want $x_{ij}(\Delta_3) > x_{ij}(0)$ for $(i, j) \in \mathcal{A}_1$ and $x_{ji}(\Delta_3) < x_{ji}(0)$ for $(j, i) \in \mathcal{A}_1$.

We now verify that the price increase in **Step 4** doesn't destroy ϵ-CS on the arcs in $\mathcal{A}_0$. Denote the new value of π_i by π'_i. We have

$$\pi'_i - \pi_j > \pi_i - \pi_j \geq f^-_{ij}(x_{ij}) - \epsilon$$

and

$$\pi'_i - \pi_j \leq \pi_i + \Delta_1 + \epsilon - \pi_j = \min_{\{v \in \delta_+(i) \,|\, (i,v) \in \mathcal{A}_0\}} \left\{ f^+_{iv}(x_{iv}) \right\} + \epsilon \leq f^+_{ij}(x_{ij}) + \epsilon$$

Similarly, ϵ-CS is preserved on an arc $(j, i) \in \mathcal{A}_0$.

Finally, we show that Steps 2–4 eventually drive the surplus of node i to 0. Assume the opposite. Then $\Delta_3 = +\infty$ in every execution of Step 4 implying that this step increases π_i by at least $\frac{\epsilon}{2}$. Hence π_i approaches infinity which is impossible as will be shown in the proof of Lemma 1 in a more general context.

2.3.2 Convergence analysis

In this section we establish convergence for **Algorithm 1**.

We refer to the pair (x, π) available at the beginning of **Step 1** (just before k is incremented) as (x^k, π^k).

The surplus of node $i \in \mathcal{N}$ at the beginning of iteration k is given by

$$s_i^k := b_i - \sum_{j \in \delta_+(i)} x_{ij}^k + \sum_{j \in \delta_-(i)} x_{ji}^k \tag{21}$$

Lemma 1 *The sequence* $\{\pi^k\}_{k=0}^{\infty}$ *generated by **Algorithm 1** converges to some point* π^*.

Proof The sequence $\{\pi_i^k\}_{k=0}^{\infty}$ is nondecreasing for all $i \in \mathcal{N}$. Hence we only need to show it is bounded above. Assume the contrary, i.e. $\pi_i^k \to \infty$ for all $i \in \mathcal{N}_1$ while π_j^k stay bounded for $j \in \mathcal{N}_0$ as $k \to \infty$. Here $\{\mathcal{N}_0, \mathcal{N}_1\}$ is a partition of $\mathcal{N}$ with $\mathcal{N}_0 \neq \emptyset$ since there is always a node with negative surplus which is never chosen in **Step 1**. Pick an arc (i, j) with $i \in \mathcal{N}_1$ and $j \in \mathcal{N}_0$. We have that $\pi_i^k - \pi_j^k \to \infty$ which together with the ϵ-CS condition

$$f_{ij}^-(x_{ij}^k) - \epsilon \leq \pi_i^k - \pi_j^k \leq f_{ij}^+(x_{ij}^k) + \epsilon$$

implies that $f_{ij}^+(x_{ij}^k) \to +\infty$. Hence for any $\bar{x}_{ij} \in \operatorname{int}\operatorname{dom} f_{ij}$ there exist an iteration number k_{ij} such that $x_{ij}^k \geq \bar{x}_{ij}$ whenever $k \geq k_{ij}$. Also, if $\bar{x}_{ij} \in \operatorname{dom} f_{ij}$ is the right endpoint of $\operatorname{dom} f_{ij}$ then $f_{ij}^-(\bar{x}_{ij}) < +\infty$ (Assumption 2.2.3) hence $f_{ij}^+(x)$ is uniformly bounded above on $\operatorname{int}\operatorname{dom} f_{ij}$ implying that $x_{ij}^k = \bar{x}_{ij}$ for $k \geq k_{ij}$.

Similarly if (j,i) is an arc with $j \in \mathcal{N}_0$, $i \in \mathcal{N}_1$ and $\bar{x}_{ji} \in \mathrm{dom} f_{ji}$ there exist an iteration number k_{ji} such that $x_{ji}^k \leq \bar{x}_{ji}$ whenever $k \geq k_{ji}$.

Now choose for each arc (i,j), $i \in \mathcal{N}_1$, $j \in \mathcal{N}_0$ the number $\bar{x}_{ij} \in \mathrm{dom} f_{ij}$ and for each arc (j,i), $j \in \mathcal{N}_0$, $i \in \mathcal{N}_1$ the number $\bar{x}_{ji} \in \mathrm{dom} f_{ji}$ such that

$$\sum_{i \in \mathcal{N}_1} b_i \quad - \sum_{\{(i,j)|i \in \mathcal{N}_1, j \in \mathcal{N}_0\}} \bar{x}_{ij} + \sum_{\{(j,i)|i \in \mathcal{N}_1, j \in \mathcal{N}_0\}} \bar{x}_{ji} \quad \leq \quad 0$$

(This is always possible since the problem is feasible.) For sufficiently large k we now have

$$\begin{aligned} \sum_{i \in \mathcal{N}_1} s_i^k \quad &= \quad \sum_{i \in \mathcal{N}_1} b_i - \sum_{\{(i,j)|i \in \mathcal{N}_1, j \in \mathcal{N}_0\}} x_{ij}^k + \sum_{\{(j,i)|i \in \mathcal{N}_1, j \in \mathcal{N}_0\}} x_{ji}^k \quad \leq \\ & \quad\quad \sum_{i \in \mathcal{N}_1} b_i - \sum_{\{(i,j)|i \in \mathcal{N}_1, j \in \mathcal{N}_0\}} \bar{x}_{ij} + \sum_{\{(j,i)|i \in \mathcal{N}_1, j \in \mathcal{N}_0\}} \bar{x}_{ji} \quad \leq \quad 0 \end{aligned}$$

which contradicts the assumption that the nodes in $\mathcal{N}_1$ are chosen infinitely many times in **Step 1**. ∎

Lemma 2 *The sequence $\{x^k\}_{k=0}^{\infty}$ generated by **Algorithm 1** converges to some $x^* \in \mathrm{dom}\left(\sum_{(i,j) \in \mathcal{A}} f_{ij}\right)$.*

Proof We start by showing that the sequence $\{x^k\}_{k=0}^{\infty}$ is bounded. First note that the quantity $\sum_{i \in \mathcal{N}} |s_i^k|$ is a nonincreasing function of k. Assume that the flow on an arc $(i,j) \in \mathcal{A}$ is unbounded (i.e. $x_{ij}^{k_l} \to \infty$ where $\{k_l\}$ is an appropriately chosen index sequence). Hence there is a cycle Y containing (i,j) such that the flows on the forward arcs of Y (denoted by Y^+) approach $+\infty$ while the flows on the backward arcs of Y (denoted by Y^-) approach $-\infty$. We assume that $\{k_l\}$ has been further thinned so that

$$x_{uw}^{k_l} \to +\infty, \quad (u,w) \in Y^+$$

$$x_{wu}^{k_l} \to -\infty, \quad (w,u) \in Y^-$$

By summing the ϵ-CS condition

$$f_{uw}^-(x_{uw}^{k_l}) - \epsilon \le \pi_u^{k_l} - \pi_w^{k_l} \le f_{uw}^+(x_{uw}^{k_l}) + \epsilon$$

along Y we get that

$$\sum_{(u,w)\in Y^+} f_{uw}^-(x_{uw}^{k_l}) \quad - \sum_{(w,u)\in Y^-} f_{wu}^+(x_{wu}^{k_l}) \le n\epsilon$$

which is a contradiction with Assumption 2.2.2 since $x_{uw}^{k_l} \to +\infty$ implies $f_{uw}^-(x_{uw}^{k_l})$ $\to +\infty$ while $x_{wu}^{k_l} \to -\infty$ implies $f_{wu}^+(x_{wu}^{k_l}) \to -\infty$.

We now show that $\{x_{ij}^k\}_{k=0}^\infty$ converges for each $(i,j) \in \mathcal{A}_1$. Since f_{ij} is proper closed strictly convex function, its conjugate f_{ij}^* is essentially smooth hence (using Assumption 2.2.2) (continuously) differentiable. We have

$$x_{ij}^k = \nabla f_{ij}^*(\pi_i^k - \pi_j^k)$$

Lemma 1 shows that $\pi_i^k - \pi_j^k \to \pi_i^* - \pi_j^*$ hence $\{x_{ij}^k\}$ converges.

To see that $\{x_{ij}^k\}_{k=0}^\infty$ converges for $(i,j) \in \mathcal{A}_0$, note that for sufficiently large k (i,j) can be chosen only in **Step 2** or only in **Step 3** but not in both of them. Hence starting with some k, $\{x_{ij}^k\}$ is monotone which together with its boundedness implies convergence.

Finally, $x_{ij}^* \in \mathrm{dom} f_{ij}$ for each $(i,j) \in \mathcal{A}$ because (x^*, π^*) satisfy ϵ-CS implying that $f_{ij}^-(x_{ij}^*)$ and $f_{ij}^+(x_{ij}^*)$ cannot both be $+\infty$ or $-\infty$. ∎

The following assumption regarding the way nodes are chosen in **Step 1** is needed in establishing convergence to a feasible point.

Assumption 2.3.1 *If $s_i^k > 0$ for some k then node i is chosen in **Step 1** in some iteration numbered higher than k.*

Lemma 3 *The sequence $\{s^k\}_{k=0}^{\infty}$ generated by **Algorithm 1** converges to* 0.

Proof Let $i \in \mathcal{N}$ be a node that is chosen infinitely many times in **Step 1** and let $\{k_l\}$ be the corresponding sequence of iteration numbers. We have

$$0 = s_i^{k_l+1} \leq s_i^{k_l+2} \leq \ldots \leq s_i^{k_{l+1}}$$

and so

$$\limsup_{k\to\infty} s_i^k = \limsup_{l\to\infty} s_i^{k_l} \leq \limsup_{l\to\infty} \|x^{k_l} - x^{k_l+1}\|_1 = 0$$

(using that the change in a node surplus doesn't exceed the sum of changes of arc flows). Hence $s_i^k \to 0$.

On the other hand, if $j \in \mathcal{N}$ is a node with $s_j^k \leq 0$ for all k we have

$$s_j^k \geq \sum_{\{u\in\mathcal{N} | s_u^k < 0\}} s_u^k \quad = \quad - \sum_{\{u\in\mathcal{N} | s_u^k \geq 0\}} s_u^k \quad \to \quad 0$$

hence $s_j^k \to 0$. ∎

The preceding three lemmas constitute the proof of the first part of the main convergence theorem. The second part is concerned with the case when

$\mathcal{A}_0 = \mathcal{A}$. In this case we define the *admissible network* w.r.t. some pair (x, π) to be $\mathcal{G}(\mathcal{N}, \hat{\mathcal{A}})$ where $(i, j) \in \hat{\mathcal{A}}$ iff

$$(i, j) \in \mathcal{A}, \;\; f_{ij}^{+}(x_{ij}) - (\pi_i - \pi_j) \leq -\tfrac{\epsilon}{2}$$

or

$$(j, i) \in \mathcal{A}, \;\; f_{ji}^{-}(x_{ji}) - (\pi_j - \pi_i) \geq \tfrac{\epsilon}{2}$$

Less formally, the admissible network contains an arc (i, j) if flow can be pushed from node i to node j according to the rules of the algorithm.

Assumption 2.3.2 (x^0, π^0) *are chosen so that the admissible network w.r.t.* (x^0, π^0) *is acyclic.*

The above assumption is trivially satisfied if (x^0, π^0) satisfy CS, for then the set of arcs of the admissible network is empty.

Theorem 4 *The sequence* $\{(x^k, \pi^k)\}_{k=0}^{\infty}$ *generated by* ***Algorithm 1*** *converges to some* (x^*, π^*) *such that* x^* *is feasible for (17) and together with* π^* *satisfies* ϵ*-CS. Furthermore, if* $\mathcal{A}_1 = \emptyset$ *the algorithm terminates finitely with a primal–dual pair satisfying the same properties.*

Proof The first part follows directly from Lemmas 1–3. For the second part we first show that if the admissible network is acyclic w.r.t. the initial choice of (x, π), it remains acyclic throughout the algorithm. Indeed, **Steps 2** and **3** obviously cannot add new arcs to the admissible network. **Step 4** increases π_i by more than $\frac{\epsilon}{2}$ and so may add some arcs originating at i to the admissible network. At the same time all arcs entering i leave the admissible network and so no cycle can be formed in it.

Now assume that the algorithm doesn't terminate. Since $\pi_i^k \to \pi_i^*$ and $\Delta > \frac{\epsilon}{2}$ in **Step 4**, we see that for sufficiently large k, $\pi_i^k = \pi_i^*$ for each $i \in \mathcal{N}$. Hence from some point on no new arcs are added to the admissible network. This together with its acyclicity guarantees that **Algorithm 1** terminates finitely. ∎

The pair (x^*, π^*) computed by **Algorithm 1** is nearly optimal. The following two lemmas (Corollary 3.1 and Proposition 3.6 from (Bertsekas et al. 1987), respectively) give the precise statement. In them, $f(x)$ and $q(\pi)$ are used to denote the primal and dual objective values of (17):

$$\begin{aligned} f(x) &:= \sum_{(i,j)\in\mathcal{A}} f_{ij}(x_{ij}), \\ q(\pi) &:= \pi^T b \; - \sum_{(i,j)\in\mathcal{A}} f_{ij}^*(\pi_i - \pi_j). \end{aligned}$$

Lemma 5 *Let $(x(\epsilon), \pi(\epsilon))$ satisfy ϵ-CS and let $x(\epsilon)$ satisfy the flow conservation constraints. Then $f(x(\epsilon)) - q(\pi(\epsilon)) \to 0$ as $\epsilon \to 0$.*

Lemma 6 *Let $(x(\epsilon), \pi(\epsilon))$ satisfy the assumptions of Lemma 5 and in addition assume that each f_{ij} is of the form (18) (i.e. finite capacity constraints). Then*

$$0 \le f(x(\epsilon)) - q(\pi(\epsilon)) \quad \le \quad \epsilon \sum_{(i,j)\in\mathcal{A}} (u_{ij} - l_{ij})$$

In the case the f_{ij} satisfy for some $d_{ij} > 0$ the inverse Lipschitz condition

$$d_{ij}(x_{ij}^1 - x_{ij}^2) \le f_{ij}^-(x_{ij}^1) - f_{ij}^+(x_{ij}^2) \qquad \text{for } \; x_{ij}^1 > x_{ij}^2$$

(for example, f_{ij} are piecewise quadratic) we can derive a bound on the distance from the point computed by **Algorithm 1** to the optimal solution set of (17).

Lemma 7 *Let (x^*, π^*) satisfy ϵ-CS and let x^* be feasible for (17). Let $\bar{x}$ be any optimal solution of (17). Then for every $(i,j) \in \mathcal{A}$*

$$d_{ij}|x^*_{ij} - \bar{x}_{ij}| \leq n\epsilon$$

Proof Fix $(i,j) \in \mathcal{A}$ and assume that $x^*_{ij} \neq \bar{x}_{ij}$. According to the Conformal Realization Theorem (Bertsekas 1991) $x^* - \bar{x}$ can be decomposed into a sum of conforming simple cycle flows. Let Y be any of the cycles containing (i,j). By summing the ϵ-CS condition (20) along Y (as in the proof of Lemma 2) we obtain

$$\sum_{(u,w)\in Y^+} f^-_{uw}(x^*_{uw}) \quad - \sum_{(w,u)\in Y^-} f^+_{wu}(x^*_{wu}) \leq n\epsilon$$

Similarly by summing the CS condition (19) along Y we get

$$\sum_{(u,w)\in Y^+} f^+_{uw}(\bar{x}_{uw}) \quad - \sum_{(w,u)\in Y^-} f^-_{wu}(\bar{x}_{wu}) \geq 0$$

Now by subtracting the above two inequalities we get

$$\sum_{(u,w)\in Y^+} \left\{ f^-_{uw}(x^*_{uw}) - f^+_{uw}(\bar{x}_{uw}) \right\} \quad - \sum_{(w,u)\in Y^-} \left\{ f^+_{wu}(x^*_{wu}) - f^-_{wu}(\bar{x}_{wu}) \right\} \leq n\epsilon \quad (22)$$

For an arc $(u,w) \in Y^+$ we have $\bar{x}_{uw} < x^*_{uw}$ while for an arc $(w,u) \in Y^-$ we have $x^*_{wu} < \bar{x}_{wu}$ and so the conclusion of the Lemma follows immediately. ∎

For large networks the distance from the computed solution to the optimal set will typically be smaller than the estimate $n\epsilon$ of the preceding lemma.

2.3.3 Computational considerations

In this section we discuss some issues related to the implementation of **Algorithm 1** and some extensions to it.

One way of enforcing Assumption 2.3.1 in practice is to consider the nodes in **Step 1** in some fixed order that includes all nodes in $\mathcal{N}$. A more economical solution is used in the linear code ϵ-RELAX (Bertsekas 1991). There the nodes that have positive surplus are kept in a queue. Nodes are "selected" from the front of the queue; if during the course of an iteration the surplus of an adjacent node (not already in the queue) becomes positive, that node is added to the end of the queue.

"Down" iterations and prevention of cycling

Algorithm 1, as described, consists only of "up" iterations (i.e. a node with a positive surplus is selected and its price may be raised). It is possible to define in a symmetrical way a "down" iteration which starts by selecting a node with negative surplus and may decrease its price. Bertsekas & Tsitsiklis (1989) has an example of the ϵ-relaxation algorithm cycling if "up" and "down" iterations are mixed arbitrarily. If we assume, however, that either the number of "up" iterations or the number of "down" iterations started at any given node is finite, cycling cannot occur. Indeed, this assumption implies that from some point in the course of the algorithm on, either only "up" or only "down" iterations are executed at any given node and our convergence proof can be used with slight modifications. The computational results in Section 4.2 demonstrate that the performance of the algorithm for nonlinear problems is significantly improved by mixing both "up" and "down" iterations.

The above assumption can be computationally checked by using a device similar to the one used in the code ϵ-RELAX-N (Bertsekas 1991). For a node i, let p_i^k be the number of "up" iterations executed at node i before iteration number k. Also let

$$S^k := \sum_{i \in \mathcal{N}} |s_i^k|$$

We can now enforce the assumption by not allowing a "down" iteration for node i at iteration number k if

$$p_i^k > C_1 + C_2(S^0 - S^k)$$

where C_1 and C_2 are any two positive constants. The quantity $C_1 + C_2(S^0 - S^k)$ is nondecreasing and bounded above and is used as a measure of the progress made by the algorithm. Alternatively, we can check whether p_i^k is greater than the number of "flow pushes" from node i to node j such that in the beginning of each "flow push" either

$$s_i \geq \mu > 0 \ \text{ and } \ s_j \leq 0$$

or

$$s_i \leq -\mu \ \text{ and } \ s_j \geq 0.$$

As long as the algorithm makes sufficient progress, "up" and "down" iterations are allowed to be mixed. If, however, too many "up" iterations are executed at a given node without achieving progress, no more "down" iterations are allowed at this node.

ϵ-Scaling

In practice the solution of problem (17) usually consists of more than one application of **Algorithm 1**. The algorithm is executed for some value of ϵ and terminated when the surplus of all nodes becomes less than some μ in magnitude (only nodes with surplus larger than μ can be selected in **Step 1**). Then ϵ and μ are reduced and the process is repeated. This is very similar to the way cost scaling algorithms operate.

2.3.4 Specialization of the algorithm for linear/quadratic problems

We now describe a specialization of **Algorithm 1** for network flow problems with mixed linear and quadratic cost functions:

$$\begin{aligned} \text{minimize} \quad & \sum_{(i,j)\in\mathcal{A}} \bar{f}_{ij}(x_{ij}) \\ \text{subject to} \quad & \sum_{j\in\delta_+(i)} x_{ij} - \sum_{j\in\delta_-(i)} x_{ji} = b_i, \quad i \in \mathcal{N} \\ & 0 \le x_{ij} \le u_{ij} \end{aligned} \tag{23}$$

where $\bar{f}_{ij}(x_{ij})$ are convex continuously differentiable quadratic functions:

$$\bar{f}_{ij}(x_{ij}) = \frac{1}{2} d_{ij} [x_{ij}]^2 + c_{ij} x_{ij}$$

with $d_{ij} \geq 0$.

Again, let $\{\mathcal{A}_0, \mathcal{A}_1\}$ be a partition of $\mathcal{A}$ such that $\mathcal{A}_1$ contains only arcs (i, j) with $d_{ij} > 0$.

Algorithm 2

Step 0 [Initialization] Start with (x, π) such that

$$x_{ij} = \begin{cases} 0 & \text{if } c_{ij} - (\pi_i - \pi_j) > \epsilon \\ u_{ij} & \text{if } d_{ij}u_{ij} + c_{ij} - (\pi_i - \pi_j) < -\epsilon \end{cases}$$

For all the remaining arcs, the initial choice of x must satisfy the capacity constraints for the arcs with linear cost,

$$\max\left\{0, \min\left\{u_{ij}, -\frac{\epsilon}{d_{ij}} - \frac{c_{ij} - (\pi_i - \pi_j)}{d_{ij}}\right\}\right\} \leq x_{ij}$$

$$\leq \min\left\{u_{ij}, \max\left\{0, \frac{\epsilon}{d_{ij}} - \frac{c_{ij} - (\pi_i - \pi_j)}{d_{ij}}\right\}\right\}$$

for the remaining arcs in $\mathcal{A}_0$ and

$$x_{ij} = \max\left\{0, \min\left\{u_{ij}, -\frac{c_{ij} - (\pi_i - \pi_j)}{d_{ij}}\right\}\right\}$$

for the arcs in $\mathcal{A}_1$.

Set $k = 0$.

Step 1 [Choose node with positive surplus] Choose a node i such that

$$s_i := b_i - \sum_{j \in \delta_+(i)} x_{ij} + \sum_{j \in \delta_-(i)} x_{ji} > 0$$

and increment k. If no such node exists, STOP.

Step 2 [Increase flow on outgoing arc in $\mathcal{A}_0$] If there is an arc $(i,j) \in \mathcal{A}_0$ such that

$$r_{ij} := d_{ij}x_{ij} + c_{ij} - (\pi_i - \pi_j) \leq -\tfrac{\epsilon}{2},$$
$$x_{ij} < u_{ij}$$

then change the flow on arc (i,j),

$$x_{ij} \leftarrow x_{ij} + \min\left\{s_i,\ u_{ij} - x_{ij},\ -\frac{r_{ij}}{d_{ij}}\right\}$$

The flow on the remaining arcs and the dual costs remain unchanged. Update s_i and if its new value is 0 go to Step 1 else return to Step 2.

Step 3 [Decrease flow on incoming arc in $\mathcal{A}_0$] If there is an arc $(j,i) \in \mathcal{A}_0$ such that

$$r_{ji} := d_{ji}x_{ji} + c_{ji} - (\pi_j - \pi_i) \geq \tfrac{\epsilon}{2},$$
$$x_{ji} > 0$$

then change the flow on arc (j,i),

$$x_{ji} \leftarrow x_{ji} - \min\left\{s_i,\ x_{ji},\ \frac{r_{ji}}{d_{ji}}\right\}$$

The flow on the remaining arcs and the dual costs remain unchanged. Update s_i and if its new value is 0 go to Step 1 else return to Step 3.

Step 4 [Raise price and change flows in $\mathcal{A}_1$] Compute

$$\Delta_1 \ := \min_{\{j\in\delta_+(i)\,|\,(i,j)\in\mathcal{A}_0,\, x_{ij}<u_{ij}\}} \left\{ c_{ij} - (\pi_i - \pi_j) \right\}$$

$$\Delta_2 \ := \min_{\{j\in\delta_-(i)\,|\,(j,i)\in\mathcal{A}_0,\, x_{ji}>0\}} \left\{ -\left(c_{ji} - (\pi_j - \pi_i)\right) \right\}$$

For each γ and for each arc $(i,j) \in \mathcal{A}_1$ let

$$x_{ij}(\gamma) = \max\left\{ 0, \min\left\{ u_{ij}, -\frac{c_{ij} - (\pi_i + \gamma - \pi_j)}{d_{ij}} \right\} \right\}$$

Similarly for each arc $(j,i) \in \mathcal{A}_1$ let

$$x_{ji}(\gamma) = \max\left\{ 0, \min\left\{ u_{ji}, -\frac{c_{ji} - (\pi_j - \pi_i - \gamma)}{d_{ji}} \right\} \right\}$$

and let Δ_3 be the value of γ such that

$$b_i - \sum_{(i,j)\in\mathcal{A}_1} x_{ij}(\gamma) - \sum_{(i,j)\in\mathcal{A}_0} x_{ij} + \sum_{(j,i)\in\mathcal{A}_1} x_{ji}(\gamma) + \sum_{(j,i)\in\mathcal{A}_0} x_{ji} = 0$$

(where $\Delta_3 = +\infty$ if a solution does not exist).

Define:

$$\Delta := \min\left\{\Delta_1 + \epsilon, \Delta_2 + \epsilon, \Delta_3\right\}$$

Change the price on node i:

$$\pi_i \leftarrow \pi_i + \Delta$$

and adjust the flow on the incident arcs in $\mathcal{A}_1$ so that CS is satisfied. The flow on all other arcs and the prices of all remaining nodes remains unchanged. Update s_i and if its new value is 0 go to Step 1 else return to Step 2.

2.4 Multipoint methods

Consider again problem (17) (we are using matrix notation for the network constraints and explicit bounds on arc flows):

$$\begin{aligned} &\min_x \sum_{l \in \mathcal{A}} f_l(x_l) \\ &Ax = b \\ &0 \le x \le u \end{aligned} \tag{24}$$

Consider an approximating problem to (24) where the nonlinear cost function f_l of any arc $l \in \mathcal{A}$ is replaced by a piecewise linear interpolant consisting of s_l segments. Next replace each such arc l by s_l arcs having the same start and end nodes as l with upper bounds equal to the lengths of the segments and costs equal to their slopes. The end result of this transformation is a linear min-cost network flow problem.

The solution to (24) is obtained by solving a sequence of linear min-cost network flow problems like the above derived from a sequence of approximating problems to (24) whose objective functions approach uniformly the original nonlinear objective. Note that the optimal objective value of any problem in the sequence provides an upper bound for the optimal value of (24); a lower bound can be obtained by solving the Lagrangian relaxation:

$$\min_{0 \leq x \leq u} \sum_{l} \{f_l(x_l) - \pi^T(A_{.l}x_l - b)\}$$

where π are dual variables associated with $Ax = b$ and obtained by solving the linear approximating problem. The algorithm terminates when the difference between the upper and the lower bound becomes smaller than some prescribed accuracy $\epsilon > 0$.

The linear network flow problems are solved using the network simplex method. Note that from a computational standpoint, it is sufficient at any given time to only store arcs corresponding to (at most) two segments (provided they bracket the current flow value) in the interpolant to f_l for each $l \in \mathcal{A}$.

The solution method for nonlinear network flow problems described above is described in (Kamesam & Meyer 1984).

Chapter 3

Parallel decomposition of the multi-commodity flow problem

3.1 Overview

In this chapter we describe a parallel decomposition method for the multi-commodity network flow problem that is based on applying a nonlinear Jacobi algorithm to the augmented Lagrangian formulation of the problem.

In section 3.2 we introduce two augmented Lagrangian formulations of the multi-commodity problem. In section 3.3 we describe the nonlinear Jacobi algorithm and discuss its convergence properties. The final section 3.4 focuses on the solution of the coordination problem arising in the decomposition. The relative merits and disadvantages of simple and complex coordination are discussed and a hybrid approach that performs very well in practice is described.

3.2 Augmented Lagrangian formulations of the MC problem

In the decomposition method that we consider in this chapter augmented Lagrangian terms are used to move the coupling constraints to the objective function of MC:

$$L_\lambda(x,p) = F(x) + \frac{1}{2\lambda}\left\{\|(\lambda(\mathcal{D}x - d) + p)_+\|_2^2 - \|p\|_2^2\right\} \tag{25}$$

The augmented Lagrangian formulation is used in the following method of multipliers:

Step 0. Let a set of multipliers p^0 be given. Set $k = 0$.

Step 1. Let x^{k+1} be the solution point of:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & L_\lambda(x, p^k) \\ \text{subject to} \quad & x \in \mathcal{B} \end{aligned} \tag{26}$$

Step 2. Update the multipliers using:

$$p^{k+1} = (p^k + \lambda(\mathcal{D}x - d))_+$$

Step 3. Update the penalty parameter λ, set $k = k + 1$ and return to **Step 1**.

If f_i are convex lower semicontinuous, the method is globally convergent to the optimal set of (1) and the sequence $\{p^k\}_{k=0}^\infty$ converges to a particular dual optimal solution. Additionally, if f_i are polyhedral, the method converges in finitely many iterations (Rockafellar 1976).

Updating λ in Step 3, although not necessary to establish convergence, can significantly improve the convergence speed of the algorithm in practice. A heuristic scheme that increases λ if the there is not a sufficient reduction in constraint violation from the previous iteration, has been successfully used.

An alternative augmented Lagrangian that is used quite frequently in practice (Gill, Murray & Wright 1981) because of its relative simplicity is the following

$$\hat{L}_\lambda(x,p) = F(x) + p^T(\mathcal{D}x - d) + \frac{\lambda}{2}\|\hat{\mathcal{D}}x - \hat{d}\|_2^2 \qquad (27)$$

Here $\hat{\mathcal{D}}$ and $\hat{d}$ denote subsets of the rows of $\mathcal{D}$ and d corresponding to a set of *active* coupling constraints (i.e. those that are satisfied as equality at an optimal point). Since the active set is not known a priori, it is approximated by an "working set" that is updated as needed as the method of multipliers progresses. The simple heuristic of adding a constraint to the working set when it becomes nearly violated and dropping it when it becomes satisfied as a strict inequality and its associated multiplier becomes zero, has proved to work well for the multi-commodity problems that we consider. We assume that such a heuristic will identify a correct set of active constraints in a finite number of iterations after which $\hat{D}$ and $\hat{d}$ remain fixed.

We have implemented both augmented Lagrangians (25) and (27). Related computational experience is reported in Chapter 4.

In the rest of this chapter we will be concerned with the solution of the minimization problem (26). The time-step k will be dropped for brevity and the expression $L_\lambda(x,p)$ will be used to denote both augmented Lagrangians (25) and (27) unless specifically stated otherwise.

3.3 The nonlinear Jacobi algorithm

A natural way to map the minimization of the augmented Lagrangian onto a parallel machine is to assign the task of updating a single commodity flow x_i to a processor i for $i = 1, 2, \ldots, K$. The parallel algorithm of this section generates a sequence $\{x^t\}_{t=1}^{\infty}$ whose accumulation points solve the minimization problem (26). In the description and the following discussion we use the shorthand notation

$$L^i(x,y) := L_\lambda((x_1, \ldots, x_{i-1}, y, x_{i+1}, \ldots, x_K), p)$$

$$\nabla L^i(x,y) := \nabla_{x_i} L_\lambda((x_1, \ldots, x_{i-1}, y, x_{i+1}, \ldots, x_K), p)$$

Nonlinear Block Jacobi Algorithm (NJA):

Step 0. Let x^0 be a given point. Set $t = 0$.

Step 1. Compute (in parallel) $x_i^{t+\frac{1}{2}} \in \mathcal{B}_i$ for $i = 1, 2, \ldots, K$ such that

$$L^i(x^t, x_i^{t+\frac{1}{2}}) \leq L_\lambda(x^t, p) \tag{28}$$

$$L^i(x^t, x_i^{t+\frac{1}{2}}) \leq \epsilon^t + \min_{y \in \mathcal{B}_i} L^i(x^t, y) \tag{29}$$

Step 2 [Coordination] Let x^{t+1} be the point that yields the best objective value among $(x_1^t, \ldots, x_i^{t+\frac{1}{2}}, \ldots, x_K^t), \;\; i = 1, 2, \ldots, K$.

Step 3. Set $t = t + 1$ and return to **Step 1**.

In Step 1, all processors in parallel approximately minimize the augmented Lagrangian in a block Jacobi fashion, i.e., variables assigned to other processors are held fixed. The newly computed points $x_i^{t+\frac{1}{2}}$ yield objective values within ϵ^t of the optimal value for the given subproblem and no worse than the starting point x^t. We assume that $\lim_{t\to\infty} \epsilon^t = 0$, i.e. the minimization problems (29) are solved with increasing accuracy as the algorithm progresses. Step 2 is a coordination step. It updates the iterate x^t one block at a time. If K is large, convergence may be slow, so alternatives are proposed in the remainder of this chapter.

3.3.1 Convergence of the method

Establishing convergence of the algorithm when coordination is performed as in Step 2 and the minimization in Step 1 is exact is an application of Theorem 3.6 of Ferris & Mangasarian (1993). The following proposition gives the formal statement of this result.

Proposition 8 *If $F \in LC^1_M(R^{n_1+n_2+\ldots+n_K})$, then each accumulation point of the sequence $\{x^t\}_{t=1}^{\infty}$ computed by the NJA with Step 2a is stationary.*

An alternative coordination method uses a *fixed-length* step in the direction determined by the points computed in Step 1:

Step 2a [Fixed step-length] Set $x^{t+1} = x^t + \tau(x^{t+\frac{1}{2}} - x^t)$.

We first prove a basic result which says, in essence, that NJA with Step 2a converges globally provided the step-size τ is taken sufficiently small.

Proposition 9 *If $F \in LC^1_M(R^{n_1+n_2+\ldots+n_K})$ is a convex function and $0 < \tau \leq K^{-1}$ then each accumulation point of the sequence $\{x^t\}_{t=1}^{\infty}$ computed by the NJA solves problem (26).*

Proof

$$
\begin{aligned}
&L_\lambda(x^{t+1}, p) - L_\lambda(x^t, p) \\
&= L_\lambda\left((\tau \sum_{i=1}^{K}(x_1^t, \ldots, x_i^{t+\frac{1}{2}}, \ldots, x_K^t) + (1-\tau K)x^t), p\right) - L_\lambda(x^t, p)
\end{aligned}
$$

$$\leq \tau \sum_{i=1}^{K} L^i(x^t, x_i^{t+\frac{1}{2}}) + (1 - \tau K) L_\lambda(x^t, p) - L_\lambda(x^t, p)$$

$$= \tau \sum_{i=1}^{K} \left\{ L^i(x^t, x_i^{t+\frac{1}{2}}) - L_\lambda(x^t, p) \right\} \leq 0 \tag{30}$$

The first inequality above follows from the convexity of L_λ and $\tau \leq K^{-1}$.

Now let $t \to \infty$. The sequence $\{L_\lambda(x^t, p)\}_{t=1}^{\infty}$ is bounded below (on a bounded set) and (by equation (30)) nonincreasing, hence converges.

Let x^* be an accumulation point of $\{x^t\}_{t=1}^{\infty}$. Let $\{t_j\}$ be a sequence of indices such that $x^{t_j} \to x^*$ and $x^{t_j + \frac{1}{2}} \to x^\circ$ (existence of an accumulation point for $\{x^{t+\frac{1}{2}}\}_{t=1}^{\infty}$ follows from the boundedness of this sequence).

Take $y \in \mathcal{B}$ and $\delta \in (0, 1)$. By equation (29)

$$-\epsilon^{t_j} \leq L^i(x^{t_j}, x_i^{t_j + \frac{1}{2}} + \delta(y_i - x_i^{t_j + \frac{1}{2}})) - L^i(x^{t_j}, x_i^{t_j + \frac{1}{2}})$$

and taking limits as $j \to \infty$, then expanding

$$0 \leq L^i(x^*, x_i^\circ + \delta(y_i - x_i^\circ)) - L^i(x^*, x_i^\circ) = \delta \nabla L^i(x^*, x_i^\circ + \tilde{\delta}(y_i - x_i^\circ))(y_i - x_i^\circ)$$

for some $\tilde{\delta} \in (0, \delta)$.

Dividing through by δ and letting $\delta \to 0$ yields

$$\nabla L^i(x^*, x_i^\circ)(y_i - x_i^\circ) \geq 0$$

Hence x_i° solves the following problem:

$$\begin{array}{ll} \underset{z}{\text{minimize}} & L^i(x^*, z) \\ \text{subject to} & z \in \mathcal{B}_i \end{array} \tag{31}$$

To see that x_i^* also solves the same problem, first take limits in (30) over the subsequence $\{t_j\}$ to get

$$0 \leq \sum_{i=1}^{K} \left\{ L^i(x^*, x_i^\circ) - L_\lambda(x^*, p) \right\}$$

Taking limits over the same subsequence in (28) and combining with above we get for each i

$$L^i(x^*, x_i^\circ) = L_\lambda(x^*, p)$$

hence x_i^* solves (31). This implies that for each i

$$\nabla_{x_i} L_\lambda(x^*, p)(y_i - x_i^*) \geq 0$$

This completes the proof that x^* solves problem (26). ∎

The NJA algorithm for minimizing the augmented Lagrangian for equality constraints and block separable objective function is analyzed in detail in (Ruszczyński 1993*a*). There the name "Diagonal Quadratic Approximation" (DQA) is used. Indeed, instead of directly minimizing the nonseparable function[1]

$$\hat{L}_\lambda(x, p) = \sum_{i=1}^{K} f_i(x_i) + p^T(\mathcal{D}x - d)$$
$$+ \frac{\lambda}{2}\|\hat{\mathcal{D}}\bar{x} - \hat{d}\|_2^2 + \lambda(x - \bar{x})^T \hat{\mathcal{D}}^T(\hat{\mathcal{D}}\bar{x} - \hat{d}) + \frac{\lambda}{2}(x - \bar{x})^T \hat{\mathcal{D}}^T \hat{\mathcal{D}}(x - \bar{x})$$

we iteratively minimize the (block) separable function

$$\sum_{i=1}^{K} f_i(x_i) + p^T(\mathcal{D}x - d)$$

[1]The quadratic penalty term in (27) has been expanded in a Taylor series around the point $\bar{x}$.

$$+\frac{\lambda}{2}\|\hat{\mathcal{D}}\bar{x}-\hat{d}\|_2^2+\lambda(x-\bar{x})^T\hat{\mathcal{D}}^T(\hat{\mathcal{D}}\bar{x}-\hat{d})+\frac{\lambda}{2}(x-\bar{x})^T\mathrm{diag}(\hat{\mathcal{D}}^T\hat{\mathcal{D}})(x-\bar{x})$$

where

$$\mathrm{diag}(\hat{\mathcal{D}}^T\hat{\mathcal{D}}) := \begin{bmatrix} \hat{D_1}^T\hat{D_1} & & 0 \\ & \ddots & \\ 0 & & \hat{D_K}^T\hat{D_K} \end{bmatrix}$$

Proposition 11 is a stronger version of Theorem 1 in (Ruszczyński 1993a). It allows for a larger step-size τ in Step 2a than the original theorem in the case when the minimization in Step 1 is carried out exactly. The constant K_1 denotes the maximum number of blocks linked by any coupling constraint. The following simple result is used in the proof of Proposition 10.

Lemma 10 *For any y*

$$\|\mathcal{D}y\|_2^2 \le K_1 \sum_{i=1}^{K} \|D_i y_i\|_2^2.$$

Proof Let $D_{i,j\cdot}$ denote the j-th row of the matrix D_i. Then

$$\|\mathcal{D}y\|_2^2 = \|\sum_{i=1}^{K} D_i y_i\|_2^2 = \sum_{j=1}^{m_0}\left[\sum_{i=1}^{K} D_{i,j\cdot} y_i\right]^2 \le \sum_{j=1}^{m_0} K_1 \sum_{i=1}^{K}[D_{i,j\cdot} y_i]^2 = K_1 \sum_{i=1}^{K} \|D_i y_i\|_2^2$$

where the inequality follows from Cauchy's inequality and the fact that for any j at most K_1 of the values $D_{i,j\cdot}y_i$, $i = 1, \dots, K$ are different from zero. ∎

Proposition 11 *If $f_i \in LC_M^1(R^{n_i})$ are convex functions and $0 < \tau < 2K_1^{-1}$, then each accumulation point of the sequence $\{x^t\}_{t=1}^{\infty}$ computed by the NJA for the augmented Lagrangian (27) when Step 1 is an exact minimization solves problem (26).*

Proof

$$
\begin{aligned}
&\hat{L}_\lambda(x^{t+1}, p) - \hat{L}_\lambda(x^t, p) \\
&= \sum_{i=1}^{K} \Big\{ f_i(x_i^t + \tau(x_i^{t+\frac{1}{2}} - x_i^t)) - f_i(x_i^t) \Big\} + \tau p^T \mathcal{D}(x^{t+\frac{1}{2}} - x^t) \\
&\quad + \frac{\lambda}{2} \|\hat{\mathcal{D}} x^t - \hat{d} + \tau \hat{\mathcal{D}}(x^{t+\frac{1}{2}} - x^t)\|_2^2 - \frac{\lambda}{2} \|\hat{\mathcal{D}} x^t - \hat{d}\|_2^2 \\
&\leq \tau \sum_{i=1}^{K} \Big\{ f_i(x_i^{t+\frac{1}{2}}) + p^T D_i x_i^{t+\frac{1}{2}} - f_i(x_i^t) - p^T D_i x_i^t \Big\} \qquad \text{[convexity of } f_i\text{]} \\
&\quad + \tau\lambda \sum_{i=1}^{K} (x_i^{t+\frac{1}{2}} - x_i^t)^T \hat{D}_i^T (\hat{\mathcal{D}} x^t - \hat{d}) + \tau^2 \frac{\lambda}{2} \|\hat{\mathcal{D}}(x^{t+\frac{1}{2}} - x^t)\|_2^2 \\
&\leq \tau \sum_{i=1}^{K} \Big\{ f_i(x_i^{t+\frac{1}{2}}) + p^T D_i x_i^{t+\frac{1}{2}} - f_i(x_i^t) - p^T D_i x_i^t \Big\} \\
&\quad + \tau \frac{\lambda}{2} \sum_{i=1}^{K} \left\{ \|\hat{D}_i x_i^{t+\frac{1}{2}} + \sum_{j \neq i} \hat{D}_j x_j^t - \hat{d}\|_2^2 - \|\hat{\mathcal{D}} x^t - \hat{d}\|_2^2 - \|\hat{D}_i (x_i^{t+\frac{1}{2}} - x_i^t)\|_2^2 \right\} \\
&\quad + \tau^2 K_1 \frac{\lambda}{2} \sum_{i=1}^{K} \|\hat{D}_i (x_i^{t+\frac{1}{2}} - x_i^t)\|_2^2 \qquad \text{[Lemma 10]} \\
&= \tau \sum_{i=1}^{K} \Big\{ \hat{L}^i(x^t, x_i^{t+\frac{1}{2}}) - \hat{L}_\lambda(x^t, p) \Big\} - \tau(1 - \tau K_1) \frac{\lambda}{2} \sum_{i=1}^{K} \|\hat{D}_i (x_i^{t+\frac{1}{2}} - x_i^t)\|_2^2 \qquad (32)
\end{aligned}
$$

When the minimization in Step 1 of the NJA is carried out exactly, i.e. $\epsilon^t = 0 \; \forall t$ we get

$$
\begin{aligned}
&\hat{L}_\lambda(x^t, p) - \hat{L}^i(x^t, x_i^{t+\frac{1}{2}}) \\
&= \Big[f_i(x_i^t) - f_i(x_i^{t+\frac{1}{2}}) \Big] + \frac{\lambda}{2} \left[\|\hat{\mathcal{D}} x^t - \hat{d}\|_2^2 - \|\hat{D}_i x_i^{t+\frac{1}{2}} + \sum_{j \neq i} \hat{D}_j x_j^t - \hat{d}\|_2^2 \right] \\
&\geq \nabla f_i(x_i^{t+\frac{1}{2}})(x_i^t - x_i^{t+\frac{1}{2}}) \\
&\quad + \frac{\lambda}{2} \left[2(\hat{D}_i x_i^{t+\frac{1}{2}} + \sum_{j \neq i} \hat{D}_j x_j^t - \hat{d})^T \hat{D}_i (x_i^t - x_i^{t+\frac{1}{2}}) + \|\hat{D}_i (x_i^t - x_i^{t+\frac{1}{2}})\|_2^2 \right] \\
&= \nabla \hat{L}^i(x^t, x_i^{t+\frac{1}{2}})(x_i^t - x_i^{t+\frac{1}{2}}) + \frac{\lambda}{2} \|\hat{D}_i (x_i^t - x_i^{t+\frac{1}{2}})\|_2^2
\end{aligned}
$$

$$\geq \frac{\lambda}{2}\|\hat{D}_i(x_i^t - x_i^{t+\frac{1}{2}})\|_2^2$$

where the last inequality follows from minimum principle necessity (Mangasarian 1969). Therefore for $0 < \tau < 2K_1^{-1}$

$$\hat{L}_\lambda(x^{t+1}, p) - \hat{L}_\lambda(x^t, p) \leq -\tau(2 - \tau K_1)\frac{\lambda}{2}\sum_{i=1}^{K}\|\hat{D}_i(x_i^{t+\frac{1}{2}} - x_i^t)\|_2^2 \leq 0 \tag{33}$$

Now let $t \to \infty$. The sequence $\{\hat{L}_\lambda(x^t, p)\}_{t=1}^{\infty}$ is bounded below (on a bounded set) and (by equation (33)) nonincreasing, hence converges.

The proof now proceeds as in Proposition 9. ∎

3.4 Solution of the coordination problem

The coordination step uses direction information computed by the subproblems to obtain the new iterate x^{t+1}. More effort spent in the coordination step usually produces better iterates and hence leads to faster convergence. On the other hand, a complex coordinator is essentially a serial step that brings down the parallel efficiency of the decomposition method.

In (De Leone et al. 1994) we focused on the coordination mechanisms employed by several of the decomposition algorithms mentioned in Chapter 1. We now consider the issue of coordination as it applies to the NJA for the augmented Lagrangian.

3.4.1 Simple coordination for NJA

Step 2a of the NJA is an example of a very simple coordinating step. The inter-processor communication in this case is limited to computing $\sum_{i=1}^{K} D_i x_i^t$ (needed by the parallel subproblems in the next round) and requires a single **vector reduce** operation in CM-5 terms (cf. Appendix B for a brief discussion). The original coordination step 2 can also be computed very efficiently.

In general, these simple coordination mechanisms lead to parallel algorithms that require relatively little inter-processor communication. However, these algorithms may also exhibit slow rates of convergence. When the number of linked blocks K_1 is large, τ must be taken very small to guarantee convergence; similarly, the NJA with Step 2 would require on the order of K steps to change all (block) components of the iterate (and thus approach a solution point of (26)). In the next section we introduce two alternative coordination schemes that lead to faster convergence of the NJA in the test problems that we consider.

3.4.2 Complex coordination for NJA

Complex coordination methods for the NJA involve the solution of a complex optimization problem. A complex coordination problem makes better use of the direction information computed by the subproblems and hence offers the possibility of improving the convergence speed of the NJA in practice. For example, the following coordination step may be substituted for Step 2 of NJA.

Step 2b [Multidimensional Search] Set $x_i^{t+1} = x_i^t + \alpha_i^*(x_i^{t+\frac{1}{2}} - x_i^t)$ where $\alpha_i^*,\ i = 1, 2, \ldots, K$ solve

$$\begin{array}{ll} \underset{\alpha_1,\ldots,\alpha_K}{\text{minimize}} & L_\lambda((x_1^t + \alpha_1(x_1^{t+\frac{1}{2}} - x_1^t), \ldots, x_K^t + \alpha_K(x_K^{t+\frac{1}{2}} - x_K^t)), p) \\ \text{subject to} & 0 \le x_i^t + \alpha_i(x_i^{t+\frac{1}{2}} - x_i^t) \le u_i, \quad i = 1, \ldots, K \end{array} \tag{34}$$

Note that $x_i^{t+\frac{1}{2}} - x_i^t \in \ker A_i$ so the equality constraints need not be included in problem (34).

Step 2b is a multi-dimensional line-search along the descent directions computed in Step 1. Its use in NJA typically leads to much faster convergence than either Step 2 or Step 2a. Its downside is that it is significantly more expensive in terms of both computation and communication. Unless the number of blocks K is small, problem (34) may be difficult to solve thus presenting a serial bottleneck for NJA. Also, in order to form (34) the products $(x_j^{t+\frac{1}{2}} - x_j^t)\hat{D}_j^T \hat{D}_i(x_i^{t+\frac{1}{2}} - x_i^t)$ need to be computed for all i and j. This adds significant computation, particularly compared to the simple coordinators Step 2 and Step 2a.

In the case of the quadratic (or linear) MC

$$f_i(x_i) := c_i^T x_i + \frac{1}{2} x_i^T Q_i x_i$$

where $Q_i,\ i = 1, 2, \ldots, K$ are positive semidefinite matrices it is possible to develop an alternative to Step 2b. It overcomes the forementioned disadvantages while attempting to preserve the fast convergence associated with use of Step 2b in NJA. In essence, we approximately solve (34) by performing one step of a nonlinear Jacobi method starting at a carefully chosen initial point.

Step 2c [Multiple Line Search] Let α^* denote the optimal solution of

$$\begin{array}{ll} \underset{\alpha}{\text{minimize}} & L_\lambda(x^t + \alpha(x^{t+\frac{1}{2}} - x^t), p) \\ \text{subject to} & 0 \le x_i^t + \alpha(x_i^{t+\frac{1}{2}} - x_i^t) \le u_i \quad i = 1, \ldots, K \end{array} \tag{35}$$

Solve (in parallel) the following K problems and let $\beta_i^*,\ i = 1, 2, \ldots, K$ denote their solution points

$$\begin{array}{ll} \underset{\beta_i}{\text{minimize}} & L^i(x^t + \alpha^*(x^{t+\frac{1}{2}} - x^t), x_i^t + \beta_i(x_i^{t+\frac{1}{2}} - x_i^t)) \\ \text{subject to} & 0 \le x_i^t + \beta_i(x_i^{t+\frac{1}{2}} - x_i^t) \le u_i \end{array} \tag{36}$$

Let x^{t+1} be the point with best objective value among the points computed in (35) and (36) and the point

$$(x_1^t + \beta_1^*(x_1^{t+\frac{1}{2}} - x_1^t), \ldots, x_K^t + \beta_K^*(x_K^{t+\frac{1}{2}} - x_K^t))$$

The first minimization in Step 2c solves (34) on the subspace $\alpha_1 = \ldots = \alpha_K$, thus providing a good starting point for one Jacobi iteration (the second minimization). The final comparison of function values guarantees sufficient decrease so that convergence of NJA with Step 2c can be guaranteed by Proposition 11.

We now show in some detail how to compute Step 2c efficiently in the case of the augmented Lagrangian for equality constraints (27). Let $u^t = \sum_{i=1}^K \hat{D}_i x_i^t$ (already available to all processors as part of the objective function of (29)) and $u^{t+\frac{1}{2}} = \sum_{i=1}^K \hat{D}_i x_i^{t+\frac{1}{2}}$ (computable by a single **vector reduce** operation). Note that the solution to the first minimization is now given by

$$\alpha^* = \left[-\frac{(u^{t+\frac{1}{2}} - u^t)^T(p + \lambda(u^t - \hat{d})) + \sum_{i=1}^K (c_i + Q_i x_i^t)^T (x_i^{t+\frac{1}{2}} - x_i^t)}{\lambda \|u^{t+\frac{1}{2}} - u^t\|_2^2 + \sum_{i=1}^K (x_i^{t+\frac{1}{2}} - x_i^t)^T Q_i (x_i^{t+\frac{1}{2}} - x_i^t)} \right]_{\#}$$

where $[\ ]_\#$ denotes projection on the appropriate box constraints.

The solution to (36) is given by

$$\beta_i^* = \left[-\frac{(x_i^{t+\frac{1}{2}} - x_i^t)^T \hat{D}_i^T (p + \lambda(\alpha^* u^{t+\frac{1}{2}} + (1-\alpha^*)u^t - \hat{d})) + (c_i + Q_i x_i^t)^T (x_i^{t+\frac{1}{2}} - x_i^t)}{\lambda \|\hat{D}_i (x_i^{t+\frac{1}{2}} - x_i^t)\|_2^2 + \sum_{i=1}^K (x_i^{t+\frac{1}{2}} - x_i^t)^T Q_i (x_i^{t+\frac{1}{2}} - x_i^t)} \right]_\#$$

One more **vector reduce** operation is needed to compute $\sum_{i=1}^K \hat{D}_i x_i^{t+1}$ in preparation for the next round of the algorithm. Thus Step 2c requires only two **vector reduce** operations as opposed to one for Step 2a. This small increase in communication is more than offset by the much increased speed of convergence observed in practical experiments. It should be noted also that Step 2c presents essentially no serial bottleneck for the NJA.

We can easily implement a coordination step for the NJA for the augmented Lagrangian (25) that is analogous to Step 2c. The pattern of inter-processor communication does not change but the optimal solutions to problems (34) and(36) can no longer be written in closed form. These problems now involve the minimization of a convex piecewise quadratic function of one variable. We compute the solution to such a problem by (a) sorting the breakpoints, and (b) scanning the list of breakpoints in order to locate the segment(s) that contain the minimizer. Nielsen & Zenios (1990) discuss other methods for this type of problem, some of which are useful in a fine-grained parallel environment.

Chapter 4

Computational results

4.1 Overview

In this chapter we present computational results obtained by using the algorithms developed in Chapters 2 and 3 to solve nonlinear min-cost network flow problems and multi-commodity network flow problems.

In section 4.2 we describe a serial implementation of the relaxation algorithm of Chapter 2 and report results on solving mixed linear/quadratic network problems generated by NETGEN (Klingman, Napier & Stutz 1974). We compare the relaxation method with an implementation of the multi-point method and study the effect of varying the percentage of arcs having nonlinear cost functions on the performance of the two respective algorithms. In section 4.3.1 we describe a parallel implementation of the nonlinear Jacobi method on the Thinking Machines CM-5 supercomputer. The test problems solved here are linear and quadratic problems generated by MNETGEN (Ali & Kennington 1977). The final section 4.3.2 focuses on a distributed implementation of the NJA on a cluster of workstations running PVM (a parallel programming environment for a collection of heterogeneous computers). We report numerical results from the solution of the PDS problems and compare them to previous results reported in the literature.

4.2 Convex min-cost network flow problems

In this section we describe computational experience using **Algorithm 2** to solve network flow test problems with mixed linear/quadratic and strictly quadratic cost functions. The code was written in C and run on a SPARCstation 20.

The quality of the computed primal-dual pair (x^*, π^*) can be measured in terms of the flow conservation constraint violation and relative accuracy of the computed objective value. For all runs reported below we had

$$\max_{i \in \mathcal{N}} \left| b_i - \sum_{j \in \delta_+(i)} x^*_{ij} + \sum_{j \in \delta_-(i)} x^*_{ji} \right| \leq 10^{-8}$$

which translates to relative feasibility violation given by

$$\max_{i \in \mathcal{N}} \left| b_i - \sum_{j \in \delta_+(i)} x^*_{ij} + \sum_{j \in \delta_-(i)} x^*_{ji} \right| / \|b\|_\infty \leq 10^{-12}$$

The computed objective value was accurate to nearly ten significant digits:

$$\left| \frac{f(x^*) - q(\pi^*)}{f(x^*)} \right| \leq 10^{-10}$$

4.2.1 The NETGEN problems

We obtained our test problems by modifying the standard NETGEN (Klingman et al. 1974) problems 1–10 and 16–25. The first ten problems are transportation problems while the rest are transshipment problems. The transshipment problems differ in number of transshipment sources and sinks, percentage of capacitated arcs and size of upper bounds. The last four problems in Table 1 were obtained by scaling the input data for the corresponding standard NETGEN problems by a factor of eight. The arc cost functions in all cases are in the range $[1, 10000]$ and the total supply in the network is at least $100,000$. We added quadratic terms to the original linear arc cost functions $c_{ij}x_{ij}$:

$$f_{ij}(x_{ij}) = r_{ij}x_{ij}^2 + c_{ij}x_{ij}.$$

By choosing the coefficients r_{ij} appropriately we generated the following three groups of test problems:

- "50% linear/50% quadratic." A quadratic term with coefficient $r_{ij} = 10.0$ was added to the cost function of 50% of the arcs (randomly chosen). In the corresponding runs we chose $\mathcal{A}_0$ to be the set of linear arcs and $\mathcal{A}_1$ to be the set of quadratic arcs.

- "50% large quadratic/50% small quadratic." A quadratic term with coefficient $r_{ij} = 10.0$ was added to half of the arc cost functions and a quadratic term with coefficient $r_{ij} = 0.001$ was added to the other half. This corresponds to a strictly convex problem with an ill-conditioned Hessian. In the algorithm we chose $\mathcal{A}_1$ to be the set of arcs with large quadratic factors and $\mathcal{A}_0$ the set of arcs with small quadratic factors.

- "100% quadratic." A quadratic term with coefficient $r_{ij} = 10.0$ was added to all arc cost functions. In the corresponding runs we chose $\mathcal{A}_0 = \emptyset$ and $\mathcal{A}_1 = \mathcal{A}$. (With these choices the algorithm is practically equivalent to the relaxation method for strictly convex problems.)

Table 1 presents the sizes of the test problems and the execution times of **Algorithm 2** in seconds for each of the three test groups. The results for the first and third groups are comparable (after taking into account the computer performance differential) to those reported in (Bertsekas et al. 1987) for similarly modified NETGEN problems. However, the method of (Bertsekas et al. 1987) had difficulties solving problems similar to "50% large quadratic/50% small quadratic," being more than five times slower for them compared to the corresponding "50% linear/50% quadratic" problems. Our method, on the other hand, solves problems in the second group in roughly the same time as corresponding problems in the first group. This shows that our algorithm is as efficient for ill-conditioned quadratic objectives as for mixed linear-quadratic objectives.

It is worth mentioning that mixing "up" and "down" iterations is very important for achieving good computational efficiency in practice. This is illustrated in Table 1 where numbers in parentheses correspond to runs where only "up"

NETGEN Number	Number of Nodes	Number of Arcs	50% linear 50% quadratic		50% large q. 50% small q.		100% quad.	
1	200	1308	0.22	(0.99)	0.24	(0.83)	0.14	(4.72)
2	200	1511	0.23		0.25		0.14	
3	200	2000	0.39		0.34		0.15	
4	200	2200	0.29		0.30		0.17	
5	200	2900	0.44	(1.46)	0.40	(1.54)	0.19	(5.77)
6	300	3174	0.37		0.41		0.24	
7	300	4519	0.84		0.71		0.29	
8	300	5169	0.91		0.85		0.37	
9	300	6075	1.16		0.98		0.48	
10	300	6320	1.27		1.11		0.50	
16	400	1306	1.25	(1.77)	0.90	(1.67)	0.28	
17	400	2443	1.10		0.80		0.25	
18	400	1306	1.46		0.83		0.28	
19	400	2443	1.05		0.84		0.26	
20	400	1416	1.0		0.82		0.29	
21	400	2836	1.49	(3.0)	1.04	(2.17)	0.27	
22	400	1416	1.16		0.81		0.29	
23	400	2836	1.29		0.87		0.27	
24	400	1382	0.85		0.98		0.28	
25	400	2676	0.75		0.73		0.27	
6S	2400	25336	7.4		7.1		9.6	
10S	2400	50535	12.4		11.5		12.1	
24S	3200	11056	7.0		7.0		5.7	
25S	3200	21408	12.6		8.7		5.5	

Table 1: Solution times for modified NETGEN problems using the ϵ-relaxation algorithm (figures in parentheses correspond to doing only "up iterations.")

NETGEN Number	Number of Nodes	Number of Arcs	50% linear 50% quadratic	50% large q. 50% small q.	100% quad.
1	200	1308	0.74	1.69	2.17
2	200	1511	0.93	1.99	2.38
3	200	2000	1.18	2.84	3.36
4	200	2200	1.25	2.61	4.01
5	200	2900	1.46	3.49	5.26
6	300	3174	2.33	5.06	7.36
7	300	4519	2.65	8.31	10.3
8	300	5169	2.76	7.76	11.1
9	300	6075	3.65	9.06	14.3
10	300	6320	3.76	9.35	13.9
16	400	1306	1.31	8.26	5.44
17	400	2443	1.57	11.5	10.1
18	400	1306	0.97	6.27	4.04
19	400	2443	1.38	8.35	7.96
20	400	1416	1.86	17.2	7.36
21	400	2836	1.57	24.7	16.2
22	400	1416	1.49	7.57	5.14
23	400	2836	1.53	10.6	10.8
24	400	1382	1.98	11.0	6.18
25	400	2676	1.51	9.95	15.8

Table 2: Solution times for modified NETGEN problems using the piecewise-linear approximation method

iterations were allowed (for some problems in the "100% quadratic" case executing only "up" iterations led to very large times not shown in the table). This should be contrasted to the observation that mixing "up" and "down" iterations does not lead to significant improvement in running time when solving linear min-cost problems using the ϵ-relaxation algorithm.

We also present the solution times for the smaller test problems using the piecewise linear approximation algorithm for separable convex network flow problems described in section 2.4. This algorithm constructs a sequence of problems having piecewise linear objective functions which converge to the original nonlinear objective and solves these problems using the network simplex method. The computed solutions in this case are feasible within machine precision but the objective function value is slightly less accurate than that computed by the relaxation method (10^{-8} vs. 10^{-10}). Except for a few of the linear-quadratic problems, the times in Table 2 are larger than the $\epsilon-$relaxation times.

Finally, our relaxation algorithm consistently outperformed the general nonlinear network optimization package LSNNO (Toint & Tuyttens 1992) on the test problems described above. LSNNO is based on a truncated Newton method. LSSNO failed to solve to optimality most of the "50% large quadratic/50% small quadratic" problems. Even when it was able to compute the optimal solution of the problem, LSSNO was one to two orders of magnitude slower than the relaxation method.

4.2.2 Effect of varying the fraction of nonlinear variables

In the previous subsection we observed that the piecewise-linear approach may be competitive with the new ϵ-relaxation method for problems in which half the arcs have linear costs, but is much less efficient for strictly quadratic problems.

This suggests that we compare more thoroughly the behavior of the two methods as the fraction of variables having nonlinear cost functions changes.

We generated versions of NETGEN Problems 5 (a transportation problem) and 17 (a transshipment problem) where the fraction of quadratic arcs was 5, 15, 25, ..., 95% of all arcs. The set of nonlinear arcs was randomly chosen but monotonically increasing. All quadratic coefficients were set to 10.0.

Figure 1 shows the running time for both methods as a function of the fraction of quadratic arcs for NETGEN Problem 7. The two are comparable as long as less than 35% of the arcs are quadratic. The running time of the multipoint method, however, increases quickly after that point while the running time of the relaxation method goes down.

Figure 2 presents the same data for Problem 17. Here the two methods are comparable as long as less than half of the arcs have nonlinear cost functions. As in the previous case, the new ϵ-relaxation method outperforms the multipoint method by a significant margin when the majority of arcs are nonlinear.

Additional computational experience has shown that the multipoint method is often superior for problems with less than 5% nonlinear arc cost functions and that the two methods are more or less competitive when the majority of arc costs are linear. On the other hand, the ϵ-relaxation method is significantly more efficient when the majority of arc cost functions are nonlinear.

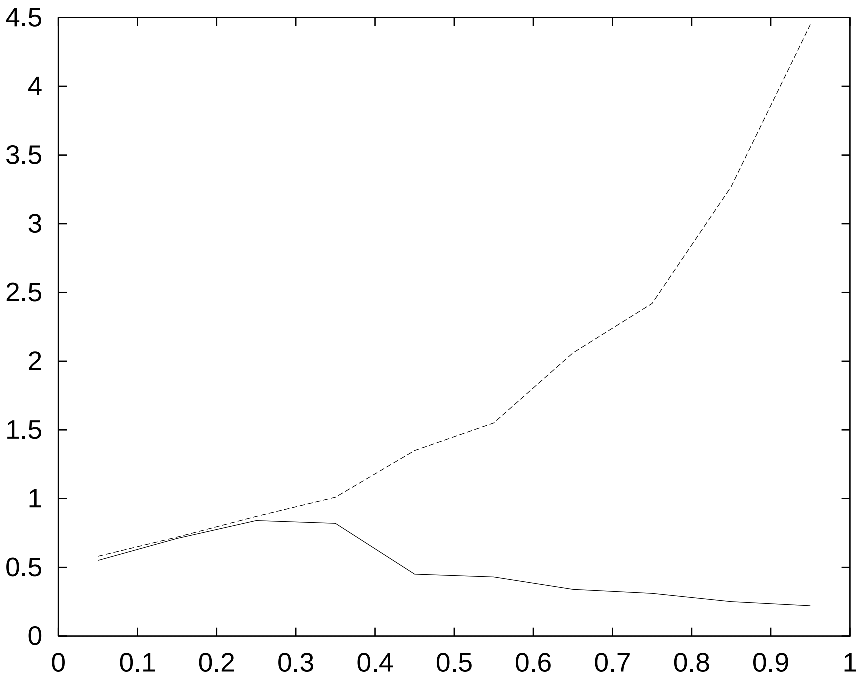

Figure 1: Solution time in seconds as a function of the fraction of nonlinear variables for the ϵ-relaxation (solid line) and multipoint methods for NETGEN Problem 5

4.3 Multi-commodity problems

We implemented the nonlinear Jacobi algorithm for the augmented Lagrangian formulation of MC on two parallel architectures—the Thinking Machines Corporation CM-5 and the Wisconsin Cluster of Workstations. We used the MNETGEN and PDS families of multicommodity problems to test our method.

4.3.1 The MNETGEN problems

We tested our algorithm on a number of multicommodity problems having from 4 up to 64 blocks. The problems were generated using MNETGEN (Ali &

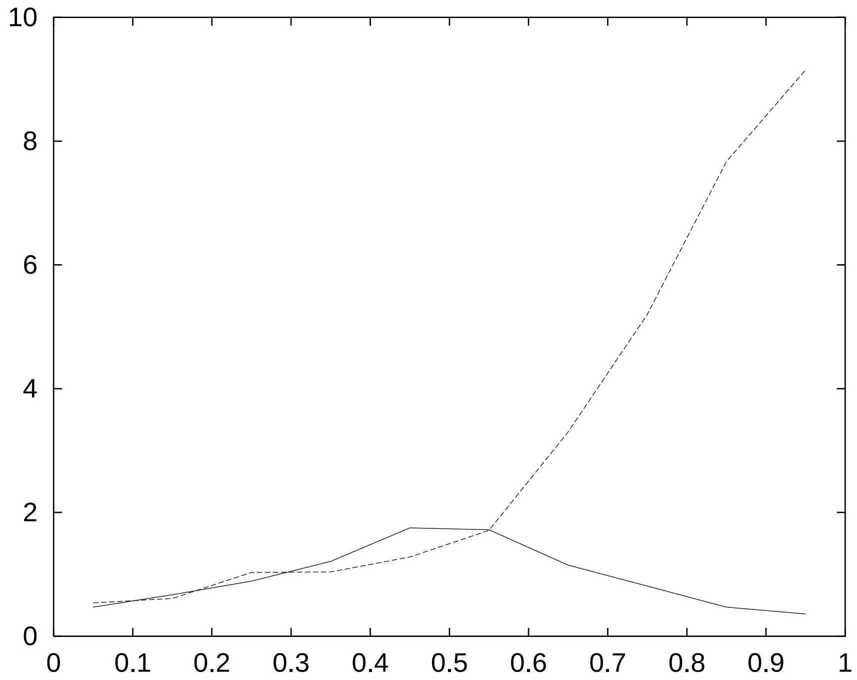

Figure 2: Solution time in seconds as a function of the fraction of nonlinear variables for the ϵ-relaxation (solid line) and multipoint methods for NETGEN Problem 17

Kennington 1977). Sizes of the underlying topological networks, numbers of coupling constraints and sizes as linear programming problems are given in Table 3. The problem name has the form M$K.n$ where K is the number of commodities and n is the number of nodes in the network. The largest problem in this suite had 39,094 constraints (excluding bounds on variables) and 74,560 variables when formulated as an LP. For these problems the coupling constraints link all blocks (i.e., $K = K_1$).

problem	#comm	#nodes	#arcs	#mutual constraints	LP size
M4.100	4	100	218	129	529×872
M8.100	8	100	229	131	931×1832
M16.100	16	100	233	157	1757×3728
M32.100	32	100	224	130	3330×7168
M64.100	64	100	220	139	6539×14080
M4.200	4	200	423	252	1052×1692
M8.200	8	200	449	277	1877×3592
M16.200	16	200	453	287	3487×7248
M32.200	32	200	456	271	6671×14592
M64.200	64	200	477	307	13107×30656
M4.300	4	300	678	409	1609×2712
M8.300	8	300	671	408	2808×5368
M16.300	16	300	661	410	5210×10576
M32.300	32	300	666	399	9999×21312
M64.300	64	300	668	393	19593×42752
M4.600	4	600	1440	945	3345×5760
M8.600	8	600	1440	1050	5850×11523
M16.600	16	600	1440	1031	10631×23041
M32.600	32	600	1440	1061	20261×46092
M64.600	64	600	1165	694	39094×74560

Table 3: MNETGEN test problems

Results for linear MNETGEN problems

We implemented the NJA with different coordination steps on a CM-5 parallel computer with 64 processors (cf. Appendix B). The multidimensional search coordination problem (Step 2b) of the NJA was solved using LSSOL (Gill, Hammarling, Murray, Saunders & Wright 1986)—a popular routine for constrained least-squares optimization.

We used the augmented Lagrangian formulation (25). The augmented Lagrangian algorithm was terminated when

$$\frac{\|p^{s+1} - p^s\|_\infty}{\|p^s\|_\infty} \leq 5 \cdot 10^{-4}$$

where p is the vector of multipliers associated with the coupling constraints. For our test problems this resulted in a relative accuracy of 10^{-6} for the optimal objective value and a maximum violation of the coupling constraints at the final solution x^* of

$$\frac{\|(\mathcal{D}x^* - d)_+\|_\infty}{\|d\|_\infty} \leq 5 \cdot 10^{-5}$$

The NJA was terminated when

$$\max_i \|\hat{D}_i(x_i^{t+\frac{1}{2}} - x_i^t)\|_\infty \leq 10^{-4}$$

or the number of iterations reached 10 (except as noted below for the method based on fixed steplengths).

The starting values for the augmented Lagrangian algorithm were

$$\begin{aligned} x^0 &= \operatorname*{argmin}_{x \in \mathcal{B}} \sum_{i=1}^{K} c_i^T x_i \\ p^0 &= 10(\mathcal{D}x^0 - d)_+ \end{aligned}$$

Table 4 shows total number of inner iterations (i.e., iterations of the NJA), total CPU time and CPU time spent in the coordination step for our algorithm with Step 2 and Step 2a. As expected, convergence of the algorithm with Step 2a was slow for both $\tau = 2K^{-1}$ and $\tau = \frac{1}{K-1}$. The table shows results for $\tau = 2K^{-1}$ which was superior to $\tau = \frac{1}{K-1}$. Ten iterations of the NJA were not sufficient to minimize the augmented Lagrangian with sufficient accuracy so higher numbers of iterations were needed, leading to relatively large running time. A comparison of table 4 with table 5 leads to the conclusion that the NJA with Step 2a is not competitive with either of the complex coordination steps. Similarly, Step 2 (interpreted as taking x^{t+1} to be the point computed by the subproblem with best objective value after Step 1 of NJA) led to very slow convergence of the algorithm for problems with 16 blocks or more. It worked reasonably well for test problems having four blocks.

problem	NJA with Step 2			NJA with Step 2a		
	iter	total time	coord. time	iter	total time	coord. time
M4.100	33	1.9		75	3.1	0.1
M8.100	66	3.9		420	18.8	0.6
M16.100	62	5.0		345	19.5	0.5
M32.100	284	27.4		510	40.9	0.8
M4.200	41	3.6		81	7.0	0.2
M8.200	135	13.1		240	23.8	0.6
M16.200	199	26.4		735	87.1	2.3
M32.200	420	91.6		495	84.7	1.2
M4.300	104	16.6		120	20.0	0.5
M8.300	200	39.32		345	61.6	1.3
M16.300	682	152.1		330	66.2	1.3
M32.300	1006	354.0		1200	422.0	5.3

Table 4: The NJA with Step 2 (uniblock updates) and Step 2a (fixed steplength)

Figure 3 graphically depicts the solution time as a function of the number of blocks for the 300-nodes problems. It can be seen that the two complex coordination schemes significantly outperform the simple coordinators for problems with more than eight blocks, and that multiple line search is the best for the largest problems.

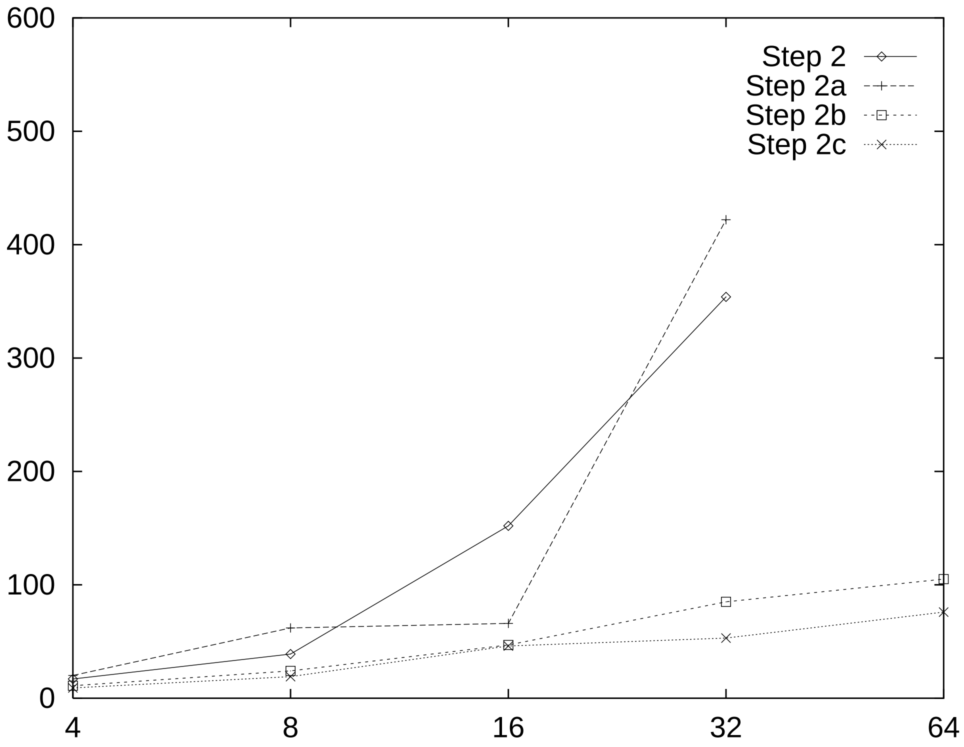

Figure 3: Solution time as a function of the number of blocks for 300-node MNETGEN problems

Table 5 shows the solution statistics for our algorithm with Step 2b (multi-dimensional search) and Step 2c (multiple line search). It can be observed that with Step 2c the number of inner iterations is comparable to Step 2b. This implies that Step 2c is able to solve the coordination problem accurately. It can also be seen that the 64 commodity problems are sometimes easier than the corresponding 32 commodity problems. This peculiar behavior of MNETGEN was previously noted by Kontogiorgis (1994).

problem	NJA with Step 2b			NJA with Step 2c		
	iter	total time	coord. time	iter	total time	coord. time
M4.100	31	2.2	0.8 (36%)	25	1.5	0.1
M8.100	51	3.9	1.1 (28%)	50	3.3	0.1
M16.100	42	4.4	1.2 (27%)	37	4.4	0.1
M32.100	80	16.1	4.0 (25%)	119	19.6	0.3
M64.100	28	4.5	2.4 (53%)	33	2.3	0.1
M4.200	18	2.8	0.8 (28%)	18	1.8	0.1
M8.200	79	9.6	1.9 (20%)	107	10.5	0.4
M16.200	76	13.8	2.8 (20%)	90	13.8	0.3
M32.200	112	30.5	7.4 (24%)	104	26.5	0.5
M64.200	97	42.5	14.2 (33%)	113	31.9	0.5
M4.300	56	10.5	1.4 (13%)	53	8.8	0.4
M8.300	110	23.9	2.9 (12%)	90	19.3	0.5
M16.300	160	46.9	8.1 (17%)	191	45.9	1.1
M32.300	197	84.8	19.2 (23%)	174	53.1	1.3
M64.300	129	105.1	35.8 (34%)	159	76.0	1.1
M4.600	20	7.2	0.8 (11%)	27	8.5	0.2
M8.600	148	61.7	3.9 (6%)	141	55.9	1.3
M16.600	218	172.8	11.8 (7%)	180	116.6	1.8
M32.600	175	276.1	34.7 (13%)	200	253.3	3.1
M64.600	69	38.1	8.9 (23%)	92	39.9	0.8

Table 5: Comparison of Step 2b (multidimensional search) and Step 2c (multiple line search)

While the time spent in the coordination phase Step 2b is a relatively small fraction of the total time for problems having up to 16 blocks, it can be seen that for problems with 32 and 64 blocks the coordination becomes expensive. It can be predicted that for problems with even higher number of blocks Step 2b will pose a significant serial bottleneck for the algorithm. Step 2c, on the other hand, takes negligible time to perform and thus the algorithm with Step 2c runs in less time than with Step 2b for problems with 32 blocks (and, in most cases, for 64 blocks as well). We can conclude that Step 2c is a viable alternative to Step 2b as a coordination mechanism for the NJA. Further, the good scalability of Step 2c makes it superior for problems with large number of commodities.

The final table 6 compares number of outer/inner (augmented Lagrangian/ nonlinear Jacobi) iterations and CPU time of our algorithm with Step 2b to those reported in (De Leone et al. 1994) for an implementation of the Dantzig–Wolfe method. Both codes use the same programming model and were run on the CM-5. Although the two algorithms can not be directly compared (the Dantzig–Wolfe method computes an exact solution within machine precision), it can be observed that CPU times for the NJA grow slower with problem size. It can be concluded that the NJA is competitive with other decomposition methods for the linear MC.

problem	NJA with Step 2b		Dantzig–Wolfe	
	iterations	CPU time	iterations	CPU time
M4.100	4/31	2	14	3
M8.100	16/51	4	14	5
M16.100	9/42	4	13	7
M4.200	3/18	3	18	11
M8.200	20/79	10	38	39
M16.200	12/76	14	39	67
M4.300	7/56	10	251	379
M8.300	11/110	24	122	298
M16.300	16/160	47	83	341

Table 6: NJA and the Dantzig–Wolfe method

Results for nonlinear MNETGEN problems

We also obtained convex quadratic versions of the MNETGEN problems by adding a proximal term to the original linear objective function:

$$f_i(x_i) := c_i^T x_i + 0.05\|x_i - x_i^0\|_2^2$$

In this case the subproblem corresponding to Step 1 of NJA is a strictly-convex ill-conditioned network flow problem. The new ϵ-relaxation method is successful in efficiently solving this type of problem. A comparison of the results for the NJA using Step 2c to the results obtained by Kontogiorgis (1994) using the Alternating Directions method briefly described in Chapter 1, is given in table 7. Again, our decomposition method compares favorably with another parallel approach for convex multicommodity programs.

problem	NJA with Step 2c		Alternating Directions
	iterations	CPU time	CPU time
M4.100	5/41	4	10
M8.100	7/51	8	15
M16.100	9/56	8	17
M4.200	5/45	5	25
M8.200	6/47	13	41
M16.200	8/65	24	34
M4.300	8/62	31	56
M8.300	9/77	42	53
M16.300	6/54	41	68

Table 7: NJA and the Alternating Directions method for quadratic problems

4.3.2 The PDS problems

The Patient Distribution System (PDS) family of problems was developed by the Airlift Mobility Command at Scott Air Force Base. Problem PDS n models the transportation of patients from a place of military conflict over a period of n days. All these problems have 11 commodities. Figure 8 gives the sizes of the networks for the problems that we used to test our method together with the sizes as linear programs (excluding flow capacity constraints).

Problem	Nodes	Arcs	Commodities	Coupling constraints	LP Size
PDS 10	1,399	4,433	11	1,169	16,558×48,763
PDS 20	2,857	10,116	11	2,447	33,874×105,728
PDS 30	4,223	15,126	11	3,491	49,944×154,998
PDS 40	5,652	20,698	11	4,672	66,844×212,859

Table 8: Characteristics of the PDS family of problems

We use the following version of the augmented Lagrangian (25):

$$L_\Lambda(x,p) = \sum_{i=1}^{K} c_i^T x_i + \frac{1}{2}\left\{\|(\Lambda(\mathcal{D}x - d) + p)_+\|^2_{\Lambda^{-1}} - \|p\|^2_{\Lambda^{-1}}\right\} \tag{37}$$

where Λ is a diagonal positive-definite matrix. In essence, we associate a separate penalty parameter with every mutual constraint. This is motivated by the significant variation in the mutual capacity values (i.e. components of d) for the PDS problems. By updating the penalty parameters separately we can compute an approximate solution that violates the tighter bundle constraints less than the looser ones. Λ_{jj} was initialized to 10 for every j and doubled in every iteration of the augmented Lagrangian algorithm in which at least a fourfold decrease in the violation of the jth bundle constraint was not achieved.

The initial values for the method of multipliers are

$$\begin{aligned} x^0 &= \operatorname*{argmin}_{x \in \mathcal{B}} \sum_{i=1}^{K} c_i^T x_i \\ p^0 &= 10(\mathcal{D}x^0 - d)_+ \end{aligned}$$

The multiple line search Step 2c is used to solve the coordination problem. The computed objective values, number of iterations (subproblem solutions) and running times on the Wisconsin COW with PVM (cf. Appendix B) for the PDS problems 10–40 are shown in Table 9. The second column is the percentage of coupling constraints that are binding at the computed optimal solution.

Problem	Optimal objective	% Binding coupl. constr.	Number of iterations	Solution time (sec)
PDS 10	$0.267271 \cdot 10^{11}$	21%	22	65
PDS 20	$0.238219 \cdot 10^{11}$	21%	66	497
PDS 30	$0.213857 \cdot 10^{11}$	22%	58	557
PDS 40	$0.188557 \cdot 10^{11}$	22%	48	858

Table 9: Solution results for the PDS problems

The objective values in Table 9 are accurate to six decimal digits. The violation of the coupling constraints at the computed solution x^* is

$$\frac{\|(\mathcal{D}x^* - d)_+\|_\infty}{\|d\|_\infty} \leq 2 \cdot 10^{-5}$$

or, componentwise,

$$\max_{1 \leq j \leq n_0} \frac{[(\mathcal{D}x^* - d)_+]_j}{d_j} \leq 2 \cdot 10^{-4}$$

An interesting question regarding the coordination Step 2c is how often the point

$$(x_1^t + \beta_1^*(x_1^{t+\frac{1}{2}} - x_1^t), \ldots, x_K^t + \beta_K^*(x_K^{t+\frac{1}{2}} - x_K^t))$$

is being selected by the multiple line search procedure. This is a measure of efficiency in the sense that no step-length computation performed by any processor, is wasted. A value of 100% would mean that the "Jacobi point" that incorporates step-length information from all the processors, is always selected. Percentages for some smaller PDS problems are given in Table 10.

Problem	"Efficiency"
PDS 1	43%
PDS 2	50%
PDS 5	56%
PDS 10	73%

Table 10: Efficiency of step-length computation for Step 2c

The PDS problems have been quite popular as a set of test problems for decomposition algorithms for MC. In Table 11 we compare our approach to other parallel decomposition methods that we briefly described in Chapter 1.

The interior point method of Schultz & Meyer (1991) uses a log-penalty barrier. Their computational experiments were conducted on a Sequent Symmetry machine. Zakeri (1995) improves on the coordination mechanism of Schultz & Meyer (1991). The Thinking Machines Corp. CM-5 was used for the computational work in this case. The smoothed exact penalty method of Pınar & Zenios (1992) was implemented on a Cray Y-MP with eight processors.

Problem	Schultz & Meyer (1991)	Pınar & Zenios (1992)	Zakeri (1995)	Current work
PDS 10	999	96	45	65
PDS 20	3,043	740	253	497
PDS 30	6,480	2,566	654	557
PDS 40	10,440	—	1,434	858

Table 11: Comparison with other parallel decomposition methods for MC

The use of different computer technology doesn't allow for a direct comparison of our method to those just mentioned. Still, we consider the Nonlinear Jacobi Algorithm for the augmented Lagrangian to be competitive with the existing methods for very large problems.

Chapter 5

Conclusions and future research

5.1 Conclusions

In this thesis we developed an efficient decomposition procedure for convex multicommodity network flow problems that is well-suited for parallel computation. We used the nonlinear Jacobi algorithm to minimize an augmented Lagrangian for the problem and embedded this procedure in a standard augmented Lagrangian algorithm. We considered several simple and complex coordination procedures for the NJA.

The method was implemented in two parallel environments—the Connection Machine CM-5 and a network of workstations running PVM. We tested our algorithm on two well-known families of multicommodity problems—the MNETGEN and the PDS problems. Our implementation compares favorably to results reported for both linear and convex multicommodity problems. We concluded that the multiple search coordinator (an approximation to multidimensional search) is superior to other coordination schemes.

We also developed a new relaxation method for convex network flow problems (the type of subproblem generated by the NJA). This algorithm maintains

ϵ-complementary slackness on a subset of the arcs and exact complementary slackness on the rest. Computational results on linear/quadratic test problems based on the standard NETGEN suite demonstrate the superiority of the method for ill-conditioned and mostly-nonlinear network flow programs.

5.2 Directions for future research

It is possible to extend our algorithm for solving multicommodity problems having much larger number of blocks than the number of available processors. In this case every processor is responsible for solving several problems (29) in Gauss–Seidel fashion. Convergence of the nonlinear Gauss–Seidel algorithm is established in (Bertsekas & Tsitsiklis 1989). We would like to combine our results with those in (Bertsekas & Tsitsiklis 1989) to prove convergence under different coordination schemes.

The parallel linear successive overrelaxation method of (Mangasarian & De Leone 1987) will usually have a better convergence speed than that of a Jacobi method. We are interested in extending the method to nonlinear objectives and performing a comparison with the nonlinear Jacobi method.

We also expect that the multiple search coordination procedure will be able to improve the convergence speed of the NJA applied to the augmented Lagrangian of the split-variable formulation of multi-stage stochastic programs (existing approaches use fixed step-length coordination with τ close to one).

This formulation typically has a large number of blocks (scenarios) so the Gauss-Seidel approach outlined above would have to be used here as well.

We would also like to study further the empirical convergence properties of the relaxation method for convex network flow programs. Dynamic adjustment of the partition $\{\mathcal{A}_0, \mathcal{A}_1\}$ may help increase the convergence speed, in particular for problems with dissimilar arc cost functions. Another issue that warrants further investigation is how to efficiently hot-start the method.

Appendix A

Totally asynchronous version of the ϵ-relaxation algorithm

An iteration of **Algorithm 1** of Chapter 2 only involves a node in the given network and its adjacent arcs. Consequently, an obvious approach to parallelization would be to assign a node (or a group of nodes) to each processor in a parallel computing environment. Iterations at several different nodes would then be able to proceed simultaneously if only iterations at independent (i.e. no two are joined by an arc) nodes were allowed to proceed concurrently. A potential problem with this approach might be limited parallel efficiency due to insufficient number of independent nodes with non-zero surplus available at any given time.

Another approach (described below) removes the node independency requirement by allowing iterations at two adjacent nodes to be executed in parallel. In this case, the two nodes may set the flow on the arc joining them to two different values. As the algorithm progresses, the difference between the two nodes' estimates of the flow approaches zero. This algorithm may be viewed as a generalization of the totally asynchronous algorithm for linear network flow problems of (Bertsekas & Tsitsiklis 1989), to nonlinear cost functions.

At time step t any node i has the following data:

- the price $\pi_i(t)$
- for any node $j \in \delta_+(i) \cup \delta_-(i)$, the price for node j at some previous time step (denoted by $\pi_j(i,t)$)
- for any arc $(i,j) \in \delta_+(i)$, its estimate of the flow $x_{ij}(i,t)$ and for any arc $(j,i) \in \delta_-(i)$, its estimate of the flow $x_{ji}(i,t)$

The surplus of node i at time t is defined to be

$$s_i(t) := b_i - \sum_{j\in\delta_+(i)} x_{ij}(i,t) + \sum_{j\in\delta_-(i)} x_{ji}(i,t)$$

At any time step t node i does one of the following:

- If $s_i(t) > 0$, executes an "up iteration" (with $\epsilon' = 0$) and updates the values of $\pi_i(t)$, $x_{ij}(i,t)$ for $(i,j) \in \delta_+(i)$ and $x_{ji}(i,t)$ for $(j,i) \in \delta_-(i)$.
- Receives messages from adjacent nodes j containing values $\pi_j(t')$ and $x_{ij}(j,t')$ for arcs $(i,j) \in \delta_+(i)$ or $x_{ji}(j,t')$ for arcs $(j,i) \in \delta_-(i)$ where $t' < t$. It uses these values to update $\pi_j(i,t)$ and $x_{ij}(i,t)$ for $(i,j) \in \delta_+(i)$ ($x_{ji}(i,t)$ for $(j,i) \in \delta_-(i)$) according to the **update rule** outlined below.

Update rule: If $\pi_j(t') < \pi_j(i,t)$, do nothing. Else set $\pi_j(i,t) \leftarrow \pi_j(t')$. For an arc $(i,j) \in \delta_+(i)$, denote the largest flow that satisfies CS together with $\pi_i(t)$ and $\pi_j(i,t)$ by $y_{ij}(i,t)$. Now if $x_{ij}(j,t') < x_{ij}(i,t)$, set $x_{ij}(i,t)$ to the projection of $y_{ij}(i,t)$ on $[x_{ij}(j,t'), x_{ij}(i,t)]$. Similarly, for an arc $(j,i) \in \delta_-(i)$, denote the largest flow that satisfies CS together with $\pi_i(t)$ and $\pi_j(i,t)$ by $y_{ji}(i,t)$. If $x_{ji}(j,t') > x_{ji}(i,t)$, set $x_{ji}(i,t)$ to the projection of $y_{ji}(i,t)$ on $[x_{ji}(i,t), x_{ji}(j,t')]$.

The initial values of (x,π) should satisfy

$$\pi_i(0) \geq \pi_i(j,0) \quad \forall j \neq i,$$

$$x_{ij}(i,0) \geq x_{ij}(j,0) \quad \forall (i,j)$$

together with the usual CS/ϵ-CS conditions.

The structure of the "up iteration", the update rule and the initial conditions imply the following properties of the iterates:

1. The sequence $\{\pi_i(t)\}_{t=0}^{\infty}$ is nondecreasing and

$$\pi_i(t) \geq \pi_i(j,t') \quad \forall t' \leq t$$

2. ϵ-CS is satisfied "locally" at each node i:

$$f_{ij}^-(x_{ij}(i,t)) - \epsilon \leq \pi_i(t) - \pi_j(i,t) \leq f_{ij}^+(x_{ij}(i,t)) + \epsilon, \quad (i,j) \in \delta_+(i)$$

$$f_{ji}^-(x_{ji}(i,t)) - \epsilon \leq \pi_j(i,t) - \pi_i(t) \leq f_{ji}^+(x_{ji}(i,t)) + \epsilon, \quad (j,i) \in \delta_-(i)$$

3. The start node estimate of the flow on an arc dominates the end node estimate:

$$x_{ij}(i,t) \geq x_{ij}(j,t) \quad \forall t$$

Proof: Let t be the first time step when the above fails. This can only happen if at time t node i received a message from j causing $x_{ij}(i,t)$ to decrease or j received a message from i causing $x_{ij}(j,t)$ to increase. Without loss of generality assume the former. Let t' be the time when the message in question was sent by j. Also let $s \leq t$ be the last time step when j received a message from i that increased $x_{ij}(j,t)$ and let s' be the time when this message was sent. We necessarily have $s > t'$ for otherwise

$$x_{ij}(i,t) \geq x_{ij}(j,t') \geq x_{ij}(j,t).$$

Now using that $x_{ij}(j,t) > x_{ij}(i,t) \geq y_{ij}(i,t)$ and the definition of $y_{ij}(i,t)$ we get

$$\begin{aligned} f_{ij}^{-}(x_{ij}(j,t)) &> \pi_i(t) - \pi_j(t') \geq \pi_i(s') - \pi_j(s) \\ &\geq f_{ij}^{-}(y_{ij}(j,s)) \geq f_{ij}^{-}(x_{ij}(j,s)) \geq f_{ij}^{-}(x_{ij}(j,t)), \end{aligned}$$

a contradiction.

4. There exists a node which never executes an "up iteration." This follows from the fact that once nonnegative, the surplus of a node remains nonnegative and from the following inequality

$$\sum_i s_i(t) = \sum_{(i,j)} \{x_{ij}(j,t) - x_{ij}(i,t)\} \leq 0 \quad \forall t$$

In order to establish convergence of the algorithm, the usual assumption is needed that information in the system never becomes "too old." Using the above Property 4, it is easy to prove in a manner similar to Lemma 1 that the node prices converge.

We now show that the flow estimates of both end nodes of each arc converge to the same value:

$$x_{ij}(i,t) \to x^*_{ij}, \qquad x_{ij}(j,t) \to x^*_{ij}.$$

Boundedness of both sequences can be established in the same way as for the sequential algorithm. Denote the largest flow value that satisfies CS together with π^*_i and π^*_j (where π^*_i and π^*_j denote the limit values for the prices of nodes i and j, respectively) by y^*_{ij}. Using that $y_{ij}(i,t)$ is the largest element of $\partial f_{ij}(\pi_i(t) - \pi_j(i,t))$ and the continuity properties of the subdifferential mapping we get

$$y_{ij}(i,t) \to y^*_{ij}, \qquad y_{ij}(j,t) \to y^*_{ij}.$$

Three cases need to be considered:

1. Both nodes i and j execute an infinite number of "up iterations" that affect their estimates of the flow on (i,j). From the logic of the "up iteration" and the update rule we get

   $$\limsup_t x_{ij}(i,t) \leq y^*_{ij} \leq \liminf_t x_{ij}(j,t)$$

   which together with Property 3 establishes the result.

2. Node i changes $x_{ij}(i,t)$ through "up iterations" only finite number of times while node j changes $x_{ij}(j,t)$ through "up iterations" infinitely many times. Then $\{x_{ij}(i,t)\}$ is nonincreasing for sufficiently large t and hence converges to some x^*_{ij}. We have

$$y^*_{ij} \leq \liminf_t x_{ij}(j,t) \leq x^*_{ij}$$

   so by the update rule

$$\liminf_t x_{ij}(j,t) = x^*_{ij}$$

   Hence $\{x_{ij}(j,t)\}$ also converges to x^*_{ij}.

3. Both nodes i and j change their respective estimates for the flow on (i,j) only finitely many times through "up iterations." Hence $\{x_{ij}(i,t)\}$ is nonincreasing while $\{x_{ij}(j,t)\}$ is nondecreasing for sufficiently large t. By the update rule they can only converge to the same value (projections of a converging sequence on a sequence of nested segments).

Finally, feasibility of the common limit points for the original problem can be established in the same way as for the serial algorithm.

Appendix B

Parallel computing environments

B.1 The Connection Machine CM-5

The Thinking Machines Corporation CM-5 is a symmetric multiprocessor that is scalable up to $16,384$ processing nodes (Thinking Machines Corporation 1991). The machine at the Computer Sciences Department of UW—Madison has 64 nodes split into two 32-node partitions that operate independently of each other. A processing node is built around a Sparc-2 CPU and additionally contains four custom vector units, 32 MB of memory and a network interface.

The nodes are connected by data, control, and diagnostic networks. The data network has a “fat-tree” topology with the nodes as leaves. Peak throughput (between adjacent leaves) is 20 MB/s. The control network is a binary tree and provides services like synchronization, scan, and reduction functions. For the latter every node contributes a data item. These data items are combined using the desired (commutative) arithmetic or logic operation and the result is returned to all nodes.

The CMMD v3.1 message-passing library provides a collection of synchronization and communication functions. Of particular interest to us is the **vector**

reduce function which takes an array in every node's memory and performs an elementwise reduction using the desired combiner operation. For example, given arrays $x^0, x^1, \ldots, x^{N-1}$ stored in the memory of nodes $0, 1, \ldots, N-1$, a **vector reduce** with combiner **add** will compute $\sum_{i=0}^{N-1} x^i$ and return it to all nodes.

We use the CMMD *hostless* programming model in which all nodes execute a copy of the same program. In our implementations no node plays a special (master) role with respect to the others; all operations are executed across all the nodes even if this leads to duplication of work.

B.2 The Wisconsin COW

The Wisconsin COW (Cluster of Workstations) is a collection of 40 SPARC-station 20 workstations. Every machine has a dual processor CPU, 64 MB of memory and runs the Solaris 2.1 operating system. The machines are connected by an Ethernet network.

We used the Parallel Virtual Machine (PVM) v3.3 package (Geist, Beguelin, Dongarra, Jiang, Manchek & Sunderam 1994) to develop a distributed version of our code. PVM makes available a rich collection of utility functions for distributed applications running on a collection of heterogeneous computers. These include creation and termination of *tasks*, synchronization, and communication. In our implementation the initial task creates $K-1$ copies of itself (i.e. once again there is not a "master" task). Communication functions used include the reduction operation (which takes a data item from each task, combines them and returns the result to a designated task) and the broadcast operation.

In our case PVM is using TCP/IP for communication. This results in very slow communication compared to the speed of computation. It is therefore extremely important (and we have made every effort) to limit as much as possible the number of calls to PVM functions that access the communication network.

Bibliography

Ali, Agha Iqbal & J. Kennington (1977), MNETGEN program documentation, Technical Report IEOR 77003, Department of Industrial Engineering and Operations Research, Southern Methodist University, Dallas, Texas 75275.

Bertsekas, Dimitri P., Patrick A. Hosein & Paul Tseng (1987), 'Relaxation methods for network flow problems with convex arc costs', *SIAM Journal on Control and Optimization* **25**(5), 1219–1243.

Bertsekas, D.P. (1991), *Linear Network Optimization:Algorithms and Codes*, The MIT Press, Cambridge, Massachusetts.

Bertsekas, D.P. & J.N. Tsitsiklis (1989), *Parallel and Distributed Computation*, Prentice Hall, Englewood Cliffs, New Jersey.

Chen, Chunhui & O.L. Mangasarian (1994), A class of smoothing functions for nonlinear and mixed complementarity problems, Technical Report 94-11, Center for Parallel Optimization, University of Wisconsin—Madison.

Collins, M., L. Cooper, R. Helgason, J. Kennington & L. LeBlanc (1978), 'Solving the pipe network analysis problem using optimization techniques', *Management Science* **24**, 747–760.

Dantzig, G.B. & P. Wolfe (1960), 'Decomposition principle for linear programs', *Operations Research* **8**, 101–111.

De Leone, R., Manlio Gaudioso & Maria Flavia Monaco (1992), Nonsmooth optimization methods for parallel decomposition of multicommodity flow problems, Technical Report 1080, Computer Sciences Dept., University of Wisconsin—Madison.

De Leone, R., R. R. Meyer, S. Kontogiorgis, A. Zakarian & G. Zakeri (1994), 'Coordination in coarse-grained decomposition', *SIAM Journal on Optimization* **4**, 777–793.

Evans, J.R. (1977), 'Some network flow models and heuristics for multiproduct production and inventory planning', *AIIE Transactions* **9**, 75–81.

Ferris, M.C. & O.L. Mangasarian (1993), Parallel variable distribution, Technical Report 1175, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin.

Geist, Al, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek & Vaidy Sunderam (1994), *PVM: Parallel Virtual Machine, A User's Guide and Tutorial for Networked Parallel Computing*, The MIT Press.

Gill, P., S. Hammarling, W. Murray, M. Saunders & M. Wright (1986), User's guide for LSSOL: A FORTRAN package for constrained linear least-squares and convex quadratic programming, Technical Report SOL 86-1, Department of Operations Research, Stanford University, Stanford, California 94305.

Gill, P.E., W. Murray & M.H. Wright (1981), *Practical Optimization*, Academic Press, London.

Glowinski, R. & A. Marrocco (1975), 'Sur l' approximation par éléments finis d' order un, et la resolution par pénalisation-dualité d' une classe de problémes de Dirichlet nonlinéaires', *Revue Française d' Automatique, Informatique et Recherche Opérationelle* **2**, 41–76.

Hu, T.C. (1966), 'Minimum cost flows in convex cost networks', *Naval Research Logistics Quarterly* **13**, 1–9.

Kamesam, P.V. & R.R. Meyer (1984), 'Multipoint methods for separable nonlinear networks', *Mathematical Programming Study* **22**, 185–205.

Kaplan, S. (1973), 'Readiness and the optimal redeployment of resources', *Naval Research Logistics Quarterly* **20**, 625–638.

Klingman, D., A. Napier & J. Stutz (1974), 'NETGEN—A program for generation of large-scale (un)capacitated assignment, transportation and minimum cost network problems', *Management Science* **20**, 814–822.

Kontogiorgis, S. (1994), Alternating Directions methods for the parallel solution of large-scale block-structured optimization problems, PhD thesis, University of Wisconsin, Madison, Wisconsin.

Korte, B. (1988), Applications of combinatorial optimization, Technical Report 88541–OR, Institute für Okonometrie und Operations Research, Bonn, Germany.

Magnanti, T.L. (1984), Models and algorithms for predicting urban traffic equilibria, *in* M.Florian, ed., 'Transportation planning models', North–Holland, Amsterdam, pp. 153–186.

Mangasarian, O. L. & R. De Leone (1987), 'Parallel successive overrelaxation methods for symmetric linear complementarity problems and linear programs', *Journal of Optimization Theory and Applications* **54**, 437–466.

Mangasarian, Olvi L. (1969), *Nonlinear Programming*, McGraw-Hill.

Medhi, D. (1987), Decomposition of structured large–scale optimization problems and parallel optimization, Technical Report 718, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin 53706. Ph.D. thesis.

Monma, C.L. & M. Segal (1982), 'A primal algorithm for finding minimum-cost flows in capacitated networks with applications', *Bell System Technical Journal* **61**, 449–468.

Nielsen, S.S. & S.A. Zenios (1990), Massively parallel algorithms for singly constrained nonlinear programs, Technical Report 90-03-01, Decision Sciences

Department, The Wharton School, Philadelphia, PA 19104. Revised March 1991.

Pınar, M.C. & S. Zenios (1992), 'Parallel decomposition of multicommodity network flows using a linear-quadratic penalty algorithm', *ORSA Journal on Computing* **4**(3), 235–249.

Rockafellar, R.T (1970), *Convex Analysis*, Princeton University Press, Princeton, NJ.

Rockafellar, R.T. (1976), 'Augmented Lagrangians and applications of the proximal point algorithm in convex programming', *Mathematics of Operations Research* **1**(2), 97–116.

Rockafellar, R.T. (1984), *Network Flows and Monotropic Optimization*, John Wiley & Sons, New York.

Rockafellar, R.T. & R.J.-B. Wets (1991), 'Scenarios and policy aggregation in optimization under uncertainty', *Mathematics of Operations Research* **10**, 119–147.

Rosa, C. & A. Ruszczyński (1994), On augmented Lagrangian decomposition methods for multistage stochastic programs, WP 94-125, IIASA, A-2361 Laxenburg, Austria.

Ruszczyński, A. (1993*a*), Augmented Lagrangian decomposition for sparse convex optimization, WP 92-75, IIASA, A-2361 Laxenburg, Austria. revised April 1993.

Ruszczyński, A. (1993*b*), 'Parallel decomposition of multistage stochastic programs', *Mathematical Programming* **58**, 201–228.

Schultz, G.L. & R.R. Meyer (1991), 'An interior point method for block angular optimization', *SIAM Journal on Optimization* **1**(4), 583–602.

Thinking Machines Corporation (1991), *The Connection Machine CM-5 Technical Summary*, Cambridge, MA.

Toint, Ph.L. & D. Tuyttens (1992), 'LSNNO: a Fortran subroutine for solving large-scale nonlinear network opimization problems', *ACM Transactions on Mathematical Software* **18**(3), 308–328.

Zakeri, G. (1995), Multi-Coordination Methods for Parallel Solution of Block-Angular Programs, PhD thesis, University of Wisconsin, Madison, Wisconsin.

Zenios, S.A., M.C. Pınar & R.S. Dembo (1991), A smooth penalty function algorithm for network–structured problems, Technical Report 90-12-05, Decision Sciences Department, The Wharton School, Philadelphia, PA 19104.